UNIVERSIDAD AUTONOMA
DE MADRID

"Autoría de Documentos Web Dinámicos Mediante Ontologías y Técnicas de Programación por Demostración"

TESIS DOCTORAL

Autor: **José Antonio Macías Iglesias**
Director: **Pablo Castells Azpilicueta**

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Junio de 2003

Memoria presentada para optar al título de Doctor en Ingeniería Informática

Esta tesis está dedicada a la memoria de mis abuelos, así como especialmente a mi familia más cercana: mis padres y hermanos. Sin su apoyo, la realización de este trabajo nunca hubiera sido posible.

Multarum hic resonat concors discordia vocum,
in te virtutum vox mage grata donat:
Religio, doctina, genus, laus belica
dulces efficiunt animi cum probiate modos;
quarum quod numeros, Princeps, expleveris,
autor perfectum numeris hoc tibi sacrat opus.

(Orlando di Lasso, S. XVI)

Agradecimientos

Aunque gran parte de este trabajo recae directamente sobre su autor, sería por otro lado injusto no valorar la ayuda y aportación de otras personas u organismos que han participado, de forma directa o indirecta, en la elaboración de esta tesis.

En primer lugar, quiero agradecer especialmente al director de la tesis su labor de supervisión durante todos estos años. Su gran experiencia en el área, junto con una metodología de trabajo meticulosa, ha hecho posible el producto final que compone esta tesis. Gracias por el esfuerzo llevado a cabo, por las buenas ideas y, en general, por haber dirigido este trabajo de investigación que tanto me ha aportado y del que tanto he aprendido en todos los aspectos.

Por otro lado quiero agradecer al Departamento de Ingeniería Informática, de la Escuela Politécnica Superior (EPS) en la Universidad Autónoma de Madrid, la oportunidad de haber realizado mi carrera investigadora en este entorno. Esta tesis nunca hubiera sido posible sin las personas y la infraestructura que componen la EPS y que, de forma directa o indirecta, tanto han aportado a este trabajo. Quiero agradecer la convivencia diaria y la ayuda prestada para llevar a cabo mi labor.

También quiero agradecer a las personas más cercanas a mí; mi familia, amigos y conocidos más cercanos, el apoyarme en todo momento y haberme inspirado en la medida de lo posible para la elaboración de este trabajo. Sin aportar ningún nombre en especial, quiero agradecer a todos los buenos consejos, la paciencia demostrada, así como la convivencia y el intercambio de ideas, lo cual siempre es reconfortante y por otro lado beneficioso para la realización de un trabajo de estas características.

Un agradecimiento especial para todas las personas que han contribuido directamente en la tesis, aportando su experiencia para la evaluación del trabajo en la experiencia con usuarios finales, así como contribuyendo desinteresadamente mediante su opinión crítica y constructiva. Gracias por los consejos y por vuestro tiempo.

Desde el punto de vista económico, el trabajo de esta tesis ha sido subvencionado a partir de tres proyectos de investigación que se solapan en el tiempo, concedidos por la Comisión Interministerial de Ciencia Y Tecnología (CICYT), proyectos número TEL-1999-0181, TIC-2001-0685-C02-01 y TIC-2002-01948.

Resumen

Actualmente, la web o tela de araña es un medio que ha pasado de contener páginas estáticas diseñadas a mano, a suministrar páginas que, son generadas dinámicamente, a partir de servidores especializados que generan información procedente de distintos soportes de almacenamiento.

Este hecho ha permitido a la web crecer en los últimos años en potencia y expresividad, pero también en complejidad y dificultad en la creación y modificación de documentos web por usuarios que son expertos en su dominio, pero no en complejos lenguajes de programación para la web.

En base a estas argumentaciones, esta tesis aporta una herramienta WYSIWYG para poner la edición de páginas web dinámicas al alcance de un autor, con poco o ningún conocimiento sobre lenguajes textuales de especificación para la web.

Las técnicas que se presentan en este trabajo de tesis están basadas en el uso de ontologías, una tendencia actualmente en auge en el campo de la web, así como en estrategias de programación mediante ejemplos, enfocadas al desarrollo por el usuario final. Este tipo de técnicas permiten interpretar intenciones implícitas de un usuario en un entorno de programación. Además, se trata de reconocer en las páginas HTML fragmentos de contenidos a partir de los cuales han sido generadas éstas. El uso de ontologías ayuda a caracterizar contenidos y separarlos de la presentación, a partir de la utilización de mecanismos de extracción y reconocimiento de estructuras en páginas HTML.

Este conjunto de técnicas se materializan en una herramienta de autor denominada DESK, junto con otras auxiliares, como PEGASUS, encargada de la representación del conocimiento del dominio y de la generación dinámica de páginas web, PERSEUS para la autoría interactiva del conocimiento del dominio, y HADES que permite la gestión de módulos así como la integración de las distintas herramientas.

Abstract

Nowadays, the World Wide Web has become a popular information distribution medium where, in the beginning, most of pages were static and hand-written to mostly turn out to supply dynamically generated web pages through specialized web servers whom provide information coming from customized storage sources.

Such a fact resulted in enhancing web power and expressivity but, on the other hand, in decreasing the ease of use for editing and creating web documents by non expert-in-programming users.

This Ph.D. thesis is aimed at supplying with a WYSIWYG authoring tool for providing the final user with an easy-to-use interface in order for enabling him/her to edit dynamically generated web pages, as well as avoiding the author from having to manipulate complex textual programming languages focused on the construction and specification of web documents.

Techniques presented in this work are focused on the use of ontologies, as a first order concern in mostly today's related web research fields, as well as on strategies for end-user development. Such techniques are intended to recognize implicit user's intents under a programming environment by means of Programming by Demonstration's mechanisms. Furthermore, explicit heuristics are exploited to detect meaningful fragments located on HTML pages, using ontologies to characterize contents and separate them from presentation's structures on web pages. The goal is to provide with feedback about page generation, enabling to build a conceptual reverse path that starts from the generated web page towards the procedural information about generation.

Most of techniques used in this thesis are put into practice by means of an authoring tool called DESK. Furthermore, others authoring tools, such as PEGASUS, for knowledge representation and dynamic web documents generation, PERSEUS for authoring interactive knowledge, and finally HADES for managing modules and integration, are taking into account to bring off the challenge of authoring dynamically generated web pages.

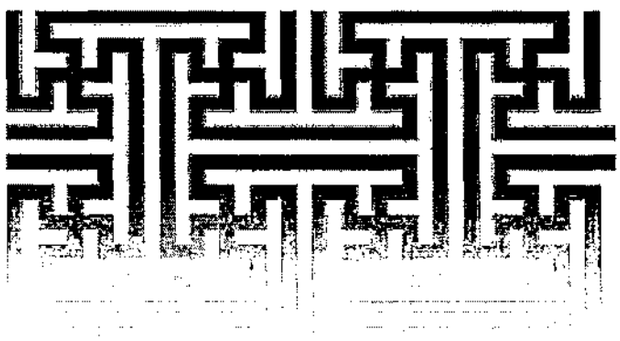
Índice de Contenidos

Índice de Figuras

Capítulo V

Índice de Tablas

Capítulo V



Capítulo I.
Introducción

En este capítulo se dará una introducción sobre la motivación de la tesis, abordando la problemática que dio lugar a su creación, junto con una descripción de la propuesta llevada a cabo para hacer frente al problema enunciado. También se comentarán los objetivos, la organización y el alcance de la tesis para continuar, en el próximo capítulo, con un estudio del trabajo relacionado con la propuesta presentada.

1.1 Definición del problema

El espectacular desarrollo de la WWW (World Wide Web) en los últimos años ha provocado un incremento en el número de empresas y particulares que utilizan este medio para la difusión de información y servicios, acorde a las necesidades de los consumidores finales de los mismos.

Sin embargo, y a pesar de su reconocido interés, la web es un medio caótico, que ha evolucionado desde la información almacenada en páginas estáticas hacia los sistemas de generación dinámica de páginas web actuales. En estos sistemas la información reside en servidores especializados, los cuales generan documentos dinámicos a partir de información almacenada en diferentes soportes.

Cabe dividir la evolución de la web en tres generaciones [DFH+00]. Por un lado, una primera generación donde predominan las páginas HTML estáticas escritas a mano. Otra segunda generación donde se incorpora el concepto de generación dinámica de documentos HTML. Y finalmente, en otra tercera generación, denominada *generación del conocimiento*, se pretende dotar de una estructura conceptual a la web bajo el paradigma de lo que se denomina *la web semántica*. En esta tercera generación, aún lejos de convertirse en una realidad, la utilización de bases de conocimiento del dominio permitiría crear servicios inteligentes, como agentes avanzados de búsqueda, agentes bursátiles más eficientes, etc.

Aunque hoy en día coexisten características de las dos primeras generaciones, se ha estimado que actualmente el 80% de los documentos a los que un usuario tiene acceso desde su navegador son generados dinámicamente [SA00].

La generación dinámica de documentos web reporta grandes ventajas a la hora de mostrar al usuario datos en tiempo real, permitiendo incluso adaptar el contenido de dicha información en función de características personales de cada usuario y de la plataforma de acceso. Sin embargo, aunque la ventaja de construir aplicaciones web que generen páginas dinámicas resulte obvia, hay que tener en cuenta los factores relacionados con el mantenimiento de este tipo de documentos web, los cuales hacen que modificar el contenido de estas páginas o cambiar su presentación no resulte una tarea fácil. Para ello se necesita un programador que conozca los lenguajes utilizados comúnmente en la composición de estas páginas, como JSP, ASP, PHP, JAVA, etc., algo que está fuera del alcance de la mayoría de los usuarios que únicamente son expertos en su dominio y no en complejos lenguajes de programación para la web. Por otro lado, han surgido en los últimos años una extensa variedad de estándares y lenguajes de especificación para la construcción de una web más estructurada, como XML, XSLT, RDF, etc. (mirar [W3C]).

En base a estas argumentaciones, sería útil idear mecanismos que permitan a un usuario no experto en programación ni en lenguajes de especificación para la web, definir y controlar la creación de documentos web dinámicos con el menor esfuerzo posible. Este problema, sencillo de exponer, es más difícil de resolver de lo que parece. La mayoría de las herramientas disponibles han sido ideadas para editar páginas web estáticas. Algunas permiten editar, parcialmente, los contenidos de páginas web dinámicas, o realizar una edición del código de generación basada en distintos mecanismos, como colorear partes del propio código o proporcionar una disposición tabular de los elementos del lenguaje. Sin embargo, no se han desarrollado herramientas que utilicen entornos de edición WYSIWYG[1] visuales, donde un usuario final pueda modificar los distintos aspectos que intervienen en la construcción de páginas web generadas dinámicamente, como los contenidos, los estilos de página y la información procedural relativa a la generación.

1.2 Objetivos

El fin último del cual arranca esta tesis es dar una solución al problema que se acaba de plantear. Para ello se abordará la edición WYSIWYG de páginas web dinámicas como eje principal de este trabajo.

Aunque esta tesis está basada en dos conceptos relacionados entre sí: la generación y la autoría de páginas web dinámicas, se narrará el trabajo personal desarrollado por el autor de la tesis: la autoría de páginas web dinámicas. Será, por tanto, el compromiso fundamental de este trabajo el de aportar una serie de técnicas que faciliten la autoría de páginas web dinámicas para sistemas de información, intentando conseguir la máxima expresividad posible poniendo como prioridad, por otro lado, la facilidad de uso del sistema propuesto. Para ello, se tomará como premisa el alejarse lo menos posible de la filosofía WYSIWYG estudiando, por otro lado, cómo de lejos puede llegarse sin abandonar esta filosofía.

Los objetivos fundamentales que se persiguen a este respecto son a) poder operar con gran libertad en la forma de representar el conocimiento, b) el diseño interactivo libre, y con poco esfuerzo, de las páginas generadas dinámicamente, independientemente de los contenidos, y c) el tratamiento de elementos no triviales de la presentación o interfaz HTML [MC01b], [MC01c]. Así pues, se investigará el desarrollo de nuevas técnicas, herramientas y paradigmas que permitan:

1. Independencia entre el diseño de la presentación y la elaboración de contenidos, a partir de un alto grado de libertad en la representación del conocimiento que será utilizado posteriormente en la autoría.

[1] What You See is What You Get: Lo que ves es lo que obtienes. Utilizado para describir interfaces de usuario de bajo nivel de abstracción donde el usuario interactúa con objetos tal y como luego aparecen en

2. Abstraer al autor de manipular directamente los distintos lenguajes de especificación existentes para la web, apostando por la facilidad de uso de la tecnología desarrollada.

3. Control para el autor de los distintos aspectos involucrados en la presentación de las páginas a editar (i.e. aspectos visuales: estructura de página, estilos, selección de controles).

4. Facilidad de integración y reutilización de contenidos y componentes de la interfaz.

5. Compatibilidad con herramientas existentes de soporte a sistemas hipermedia de generación dinámica de documentos web.

1.3 Solución propuesta

Para hacer frente a la problemática descrita en apartados anteriores, se ha desarrollado un sistema de autoría de páginas web dinámicas que permite cumplir los objetivos propuestos en el Apartado 1.2. Las herramientas diseñadas tienen distintos objetivos, siendo el foco central de este trabajo la herramienta de autor DESK, que se encarga fundamentalmente del proceso de autoría. DESK complementa otras herramientas que sirven de base al trabajo desarrollado: por un lado PERSEUS usado para autoría interactiva de la ontología y bases de conocimiento, HADES encargado de la integración de los distintos módulos y herramientas, y finalmente PEGASUS como sistema de representación del conocimiento y generación de páginas web dinámicas. Esta última herramienta sirve como base para la generación de documentos web hipermedia que DESK procesará para llevar a cabo el proceso de autoría. A continuación se apunta una breve introducción de cada una de las herramientas mencionadas anteriormente.

1.3.1 Autoría mediante ejemplos

DESK [MC02a], [MC02b], [MC03a], como entorno de edición WYSIWYG para páginas web dinámicas, permite al usuario final editar una página HTML generada dinámicamente por PEGASUS, infiriendo posteriormente los cambios en los modelos internos de PEGASUS, es decir, los modelos subyacentes de la presentación y del dominio, cambiando finalmente la forma en la que se generaran las futuras páginas web de forma persistente. De esta forma, y ordenando descendentemente de mayor a menor generalidad, DESK se puede ver como:

- Un sistema de desarrollo de interfaces de usuario mediante ejemplos

---

la salida.

- Una herramienta de autor de páginas web dinámicas.
- Una herramienta de autor para PEGASUS.

DESK utiliza técnicas basadas en el paradigma de la *Programación por Demostración* [Cyp93], [ACM00]. La herramienta de autor monitoriza al usuario durante toda la interacción, construyendo un modelo estructurado de la interacción con el usuario. Posteriormente procesa este modelo para extraer información sobre los cambios realizados por el usuario, y poder inferir automáticamente las modificaciones establecidas, evitando así que el usuario tenga que manipular los lenguajes de modelado interno de PEGASUS.

Siguiendo bajo este paradigma, y teniendo siempre en mente la automaticidad y la facilidad de uso como directrices principales, DESK incorpora un agente de asistencia para poder inferir cambios automáticamente mediante la detección de patrones de iteración a partir de las acciones acometidas por el usuario. Esto permite automatizar acciones que pueden resultar pesadas de llevar a cabo por parte del usuario final, como la transformación de elementos de la presentación (p.e. una lista de elementos en una tabla), conservando internamente las referencias semánticas con el modelo del conocimiento.

Para hacer efectivo este procesamiento, DESK consta de dos partes bien diferenciadas: una aplicación local y otra remota que corre en un servidor. La herramienta local se encargará de localizar contexto sintáctico sobre los cambios llevados a cabo por el usuario, y la herramienta remota de localizar contexto semántico para caracterizar los cambios en los modelos subyacentes y, finalmente, hacer persistentes los cambios que afectarán a la futura generación de páginas web en PEGASUS.

1.3.2 Herramientas complementarias

PEGASUS [MC01c], [MC01d], [CM02b] es un sistema de representación del conocimiento y generación automática de páginas web dinámicas, que utiliza ontologías del dominio a la medida para la descripción y estructuración conceptual de la información. PEGASUS proporciona un lenguaje de representación de las ontologías del dominio y bases de conocimiento. Una ontología describe conceptos de un dominio determinado, y consiste básicamente en una jerarquía de categorías, con propiedades y relaciones entre ellas. El vocabulario definido por la ontología se utiliza para especificar entidades del dominio (instancias de las categorías) y relacionarlas unas con otras, estructurando el conocimiento en forma de red semántica. PEGASUS además aporta un lenguaje de representación de un modelo abstracto explícito de la presentación, que determina la composición y apariencia de las páginas a generar. Estas presentaciones no se asociarán a unidades de conocimiento concretas, sino a clases y relaciones de la

ontología, es decir, a *categorías* de conocimiento. Las plantillas de presentación de PEGASUS permiten incluir referencias y condiciones sobre el modelo del usuario y de la plataforma, así mismo la utilización de un modelo abstracto de la presentación, que supone la separación entre presentación y contenidos, favorece la reutilización y consistencia en las presentaciones web diseñadas.

El motor de generación de páginas web de PEGASUS permite visualizar entidades de la base de conocimiento y navegar a través de ellas. Además, PEGASUS emplea mecanismos para adaptar la navegación a características del usuario y de la plataforma, aunque DESK apenas utiliza realmente esta característica.

PERSEUS [MC01e] es una herramienta de autor que permite diseñar las distintas formas de representación del conocimiento utilizadas en PEGASUS. Es decir, permite la creación de ontologías y la instanciación de una red semántica, sirviendo como herramienta de autor para poder desarrollar el conocimiento de forma sencilla por distintos usuarios. La interacción con PERSEUS se lleva a cabo bajo un entorno orientado a objetos, donde un diseñador puede componer conocimiento del dominio a medida bajo los paradigmas de reusabilidad y facilidad de uso. La metodología de trabajo de PERSEUS se lleva a cabo en tres pasos, 1) la creación de clases que formarán la ontología del dominio, 2) la instanciación de clases para crear la red semántica de objetos del dominio y, 3) la generación de un fichero XML que contendrá el modelo del conocimiento completo a utilizar posteriormente por PEGASUS. Mediante PERSEUS el diseñador puede navegar por la información del dominio modificando y cambiando los valores que estime oportuno, reconociendo el sistema los valores de clases y atributos ya definidos para incorporar alta funcionalidad, como la herencia automática de información, a partir de una clase, de los distintos atributos y relaciones que forman la ontología del dominio.

Finalmente **HADES** [MC01e], como portal adaptativo, permitirá integrar cada una de las herramientas anteriores, permitiendo un flujo correcto de la información entre las entradas y las salidas de las herramientas. El objetivo, por tanto, es obtener una interfaz basada en la red que pueda detectar la interacción con el usuario y compartir algunos datos comunes, como un modelo del usuario y de la plataforma utilizado por las restantes herramientas para facilitar el acceso a ciertas opciones o adecuar la presentación, generada por PEGASUS, al usuario que está interactuando con ella.

1.4 Dificultades

El desarrollo de un entorno WYSIWYG para la edición de páginas web dinámicas es un problema intrínsecamente difícil. En primer lugar supone proporcionar al diseñador una vista concreta de algo que en tiempo de diseño no existe. La estrategia propuesta a este respecto en esta tesis consiste en mostrar al diseñador un ejemplo del tipo de página que se generará en tiempo de ejecución. A partir de ahí, y de lo que el

diseñador haga con este ejemplo, el sistema debe ser capaz de mantener un vínculo de ida y vuelta entre el ejemplo y su fuente: los modelos, estructuras y procedimientos genéricos de generación. Desde una página web generada, el entorno tiene que ser capaz de reconstruir la información a partir de la cual la página ha sido construida dinámicamente.

Para hacer este problema tratable, la tesis presupone la utilización de un sistema de generación de páginas concreto (que se describe en el Capítulo III) que facilita esta labor, describiendo y dividiendo las fuentes y procedimientos de generación en modelos explícitos, susceptibles de ser analizados por el sistema de autoría. Aún así, el sistema tiene que ser capaz de *ver* a través de las transformaciones sufridas por las piezas originales (por ejemplo, una celda de una tabla HTML que procede de una determinada estructura de datos y un modelo de presentación iterativa), y afrontar la ambigüedad derivada del hecho de que distintas variaciones en las fuentes pueden producir el mismo resultado.

Por otra parte, el sistema tiene que traducir las acciones del usuario sobre los ejemplos en acciones sobre la fuente de los mismos. Esta correspondencia conlleva un acto de interpretación de la intención del usuario, no exenta, de nuevo, de ambigüedad. Cuando el diseñador edita un párrafo, cambia un estilo, reordena una lista, o modifica una celda de una tabla, el sistema debe determinar qué representan para el diseñador los elementos manipulados, y decidir cuál es el nivel de generalización adecuado para las acciones. Generalizar una acción significa que lo que el diseñador haga con un caso concreto habrá de tener efecto sobre un conjunto de casos. Decidir automáticamente si este conjunto debe ser amplio o restringido, abarcar todos los casos, sólo algunos, cuáles, y con qué criterio, es un problema complejo. Este problema no tiene en general una solución que garantice el acierto, sino que solamente admite una aproximación heurística.

En esta tesis, como premisa, se prescinde intencionadamente de aumentar el entorno de edición con elementos y abstracciones ajenos a un editor estándar de HTML. En estas condiciones el sistema dispone de muy poca información explícita y mucha información implícita. A partir de estos elementos, el sistema debe ingeniárselas para derivar la máxima información posible, atribuirle significado, y llegar tan lejos cómo sea posible con tan sólo esta información. En ese sentido este trabajo pretende contribuir a comprender los límites esperables de una estrategia de este tipo [MC03b].

La aproximación aquí propuesta consistirá en ayudar a la herramienta dándole acceso a una representación semántica explícita de las fuentes de generación de los documentos, como se describirá en los próximos capítulos.

1.5 Principales aportaciones

A nivel general, los objetivos planteados en esta tesis suponen una aportación a distintos campos de investigación, que se puede resumir en:

- Dotar de herramientas de autor interactivas a un sistema de visualización y navegación hipermedia sobre bases de conocimiento basado en ontologías y páginas dinámicas, abarcando todos los aspectos de la autoría.

- El desarrollo de nuevas técnicas de extracción de información y análisis de tablas en documentos HTML.

- El desarrollo de nuevas técnicas de programación mediante ejemplos para la edición de documentos HTML dinámicos, basadas en el análisis de trazas y la detección de patrones de acción iterativos.

- La utilización explícita de ontologías del dominio, en la línea propuesta por la web semántica, para mejorar la extracción de información en documentos web.

- La utilización de las técnicas de extracción de información desarrolladas, y otras tomadas del estado del arte, en un sistema de programación mediante ejemplos.

- La utilización de conocimiento explícito del dominio de aplicación (ontologías del dominio) para la mejora de las técnicas de programación mediante ejemplos.

- Demostrar la utilidad del paradigma de las interfaces basadas en modelos para el desarrollo de herramientas auxiliares inteligentes. En este caso, para el desarrollo de técnicas avanzadas de desarrollo por el usuario final.

Los cuatro últimos puntos se pueden resumir como la confluencia y utilidad recíproca de técnicas tomadas de cuatro áreas distintas y hasta ahora poco interconectadas: la programación mediante ejemplos, la extracción de información, las interfaces de usuario basadas en modelos, y las ontologías de la web semántica.

1.6 Limitaciones

La facilidad de uso es una de las premisas fundamentales del trabajo desarrollado en esta tesis. No obstante, este trabajo no pretende dar una herramienta universal para cualquier usuario. En definitiva, el trabajo aquí planteado apunta a grandes rasgos a tres tipos de autores: administradores del entorno, que administraran el portal HADES y los módulos como nexo de unión entre las distintas herramientas; diseñadores de la ontología, que utilizarán la herramienta PEGASUS; y expertos del dominio, que utilizarán DESK para editar la presentación de una forma más asequible.

Estos usuarios se enfrentarán a un nivel diferente de dificultad cada uno en sus tareas y, como se verá durante el desarrollo de esta tesis, manipularán las distintas herramientas presentadas en la solución del problema.

Por otro lado, esta tesis aborda sólo una parte de todo lo que puede implicar la generación dinámica de hipermedia. En este sentido, quedan intencionadamente fuera de los objetivos de esta tesis los siguientes aspectos:

a. Actualización automática del modelo del usuario, problema que ha sido objeto de trabajos y tesis doctorales por sí solo [AG99], [Enc97a]. Aunque se utiliza un sistema de generación dinámica de documentos hipermedia adaptativo (PEGASUS), en la edición con DESK se hace un tratamiento muy limitado del modelo del usuario, considerándose éste un aspecto que queda fuera de las posibilidades WYSIWYG de la herramienta.

b. Tratamiento del diálogo con el usuario. En esta tesis se trata información estática de una base de conocimiento, no servicios. Queda fuera del alcance de la tesis la definición de ontologías de *procedimientos*, para una descripción estructurada de los servicios a proporcionar así como de las necesidades de los usuarios. En definitiva, en esta tesis se aborda el aspecto de la presentación y no de la interacción o comportamiento del usuario en la interfaz, en comparación con los antecedentes existentes en el campo de interfaces de usuario.

c. Guiado del proceso de diseño. Las técnicas a desarrollar en esta tesis proporcionarán al diseñador una gran capacidad expresiva que, mal empleada, puede resultar en errores e inconsistencias. Por lo que respecta a esta tesis, esta responsabilidad se dejará en manos de los diseñadores.

d. Creación desde cero de una interfaz web. DESK es más adecuado para retocar un diseño de página dinámica existente que para crearlo desde cero. Esto se debe a que DESK no permite añadir nuevas unidades de conocimiento, es decir, DESK no puede crear nuevas instancias del dominio como objetos en la plantilla de la presentación, sino modificar las que ya existen.

e. Utilización de lenguajes embebidos dentro del documento web. El tratamiento que hace DESK cliente sobre la página web se lleva a cabo únicamente a partir del HTML generado por PEGASUS. No se consideran, por tanto, controles activos ni código embebido procedente de lenguajes como, por ejemplo, JavaScript.

El trabajo de esta tesis es complementario a los problemas que se acaban de enumerar, siendo estos una excelente vía para la continuación del trabajo en el futuro.

1.7 Estructura del trabajo presentado

De acuerdo con los objetivos establecidos, así como con la solución aportada en esta tesis, se establecerá una estrategia para la exposición del trabajo.

En un primer lugar, y continuando con lo narrado en este Capítulo I, se abordará en el Capítulo II el trabajo relacionado con las técnicas y herramientas presentadas, analizando las características que integran cada uno de los hitos propuestos y presentando otros sistemas que utilizan la filosofía implícita en el trabajo desarrollado.

En el Capítulo III se abordará la generación de páginas web dinámicas a partir de algunas de las herramientas comentadas en esta propuesta, enumerando las técnicas principales utilizadas en la generación y poniendo de relieve los objetivos propuestos para la adecuación del trabajo a partir de cual, en el Capítulo IV, se entrará de lleno con las técnicas utilizadas para la autoría de páginas web dinámicas, en base a lo expuesto en el capítulo anterior. El Capítulo IV es el eje central de la tesis, y todo lo descrito en este capítulo será puesto en práctica mediante un escenario de uso en la autoría de documentos web.

Finalmente, en el Capítulo V se aportará una evaluación empírica a partir de una experiencia realizada con usuarios sobre la autoría de una interfaz web. Por otra parte se procederá a comparar la propuesta presentada con otras herramientas y sistemas relacionados, para acabar dando unas conclusiones y discutiendo las aportaciones principales de este trabajo. En ese mismo capítulo se propondrán, finalmente, futuras líneas de trabajo relacionadas con el ámbito de esta tesis.

Al final, después del Anexo, se proporciona una relación de la bibliografía utilizada en este trabajo, que incluye las comunicaciones más importantes surgidas a raíz de esta tesis en distintos congresos y medios de difusión, tanto nacionales como internacionales.

1.8 Terminología empleada

Dado que en esta tesis confluyen distintos campos, cada uno con una terminología propia, es conveniente aclarar el sentido de algunos términos que se utilizarán a lo largo de toda la tesis, y en particular señalar las equivalencias entre ellos en este contexto. Estos términos son los que a continuación se detallan:

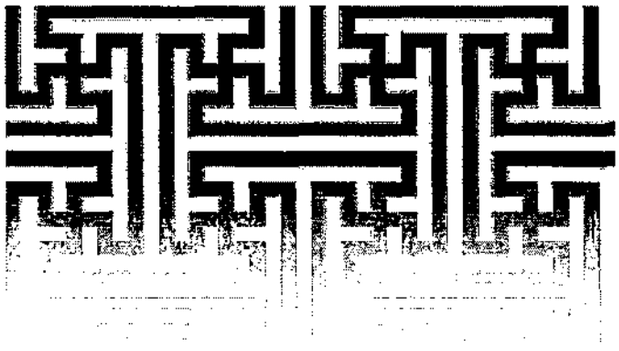
- Interfaz de usuario ≡ documento HTML. Por lo general una página web se puede ver como un documento, pero en este caso también es una interfaz de usuario. HTML y los navegadores serán en este trabajo la plataforma de soporte para las interfaces.

- Widget ≡ control HTML. El término widget se refiere comúnmente a componentes gráficos de la interfaz de usuario con aspecto y comportamiento propio, como botones, listas de selección, etc. En la línea de lo indicado en el punto anterior, aquí los widgets serán controles HTML.

- Modelo del dominio ≡ontología (vocabulario) + base de conocimiento. En el escenario en el que se sitúa este trabajo el usuario final interactúa desde la web con una base de conocimiento diseñada de acuerdo con una ontología del dominio que define los términos (clases y relaciones) utilizables para construir la base de conocimiento, según la filosofía de la web semántica. La base de conocimiento consiste en una red semántica de instancias de la ontología. Aunque en sentido estricto el modelo del dominio incluiría sólo la ontología, a menudo con este término incluiremos también por extensión la base de conocimiento. En cierto modo el modelo del dominio es asimilable a (o incluye) lo que en otros contextos del campo de las interfaces de usuario se denominaría modelo de datos (aquí se trataría de un modelo de datos enriquecido).

- Presentación ≡diseño de página. Muy a grandes rasgos y con variabilidad según los autores y el enfoque, es frecuente en la literatura sobre el desarrollo de interfaces de usuario la división de las partes de una interfaz en presentación, comportamiento, datos, funcionalidad, tareas del usuario, entre otros aspectos, con estos u otros nombres. Por presentación entenderemos aquí aquélla parte de una interfaz de usuario que incluye su aspecto visual en pantalla, la forma en la que se representan visualmente los elementos de una base de conocimiento, la selección de controles HTML, los estilos, y la disposición espacial (*layout*). En nuestro contexto, "presentación" se puede asimilar pues a "diseño de página".

- Aplicación ≡sistema de presentación de información. Como ya se ha indicado respecto del alcance de esta tesis, en cuanto al producto final de las herramientas (no así en las herramientas mismas), las técnicas desarrolladas se centran en la presentación de información, y no incluyen el tratamiento con plena generalidad de la componente interactiva de una interfaz de usuario, es decir, en las interfaces sólo se considera una forma restringida de *input* del usuario. Dicho de otra forma, en las interfaces creadas, tratadas o modificadas por las herramientas, no se contempla más acción del usuario que la interacción propia de la navegación (atravesar hiperenlaces), ni más acciones del sistema que acceder a la base de conocimiento, generar nuevas páginas y enviarlas al cliente. Por lo tanto, cuando se hable de aplicación, en definitiva se estará considerando básicamente la presentación de información, obviando (aunque no excluyendo) la posible ejecución de servicios en el servidor.



Capítulo II.
Trabajo Relacionado

En este capítulo se narrará el estado del arte relacionado con las técnicas aportadas en esta tesis para la generación y autoría de páginas web dinámicas, haciendo un recorrido por los distintos paradigmas relacionados y describiendo diferentes sistemas y herramientas involucradas en ellos.

2.1 Introducción

A partir del problema planteado, se describirá en este capítulo el trabajo relacionado en función de las técnicas empleadas en la tesis.

DESK lleva a cabo la extracción de conocimiento (i.e. fragmentos de contexto y controles de la interfaz) a partir de la página HTML generada por PEGASUS. Es por esto que uno de los temas fundamentales relacionados con este trabajo es el de los *Sistemas de Extracción del Conocimiento en Páginas Web*.

DESK es una herramienta WYSIWYG donde se considera un usuario interactuando en la autoría del documento HTML, y donde se intenta saber en todo momento qué acciones realiza el usuario y sobre qué partes del documento HTML actúa, para que finalmente esos cambios sean llevados a cabo modificando la forma de generar las páginas en el futuro. Esto atañe directamente a la *Programación por Demostración*, un campo estrechamente relacionado con la temática de la tesis y que permite aportar una visión más adecuada al problema, a partir de técnicas de desarrollo por el usuario final.

Un rasgo distintivo en este trabajo de tesis respecto a la *Programación por Demostración* y la *Extracción del Conocimiento en Páginas Web*, es el uso de un modelo del dominio. Esto permite a DESK caracterizar fragmentos de un documento, desambiguando las acciones del usuario, para inferir y modificar información de los modelos subyacentes de PEGASUS (i.e. modelos del dominio y de la presentación) encargados de la generación de la interfaz web. Así pues, el uso de modelos explícitos de las componentes de una aplicación es un aspecto clave en DESK y PEGASUS. Este tipo de estrategia ha sido utilizada en el campo de la Interacción Persona-Ordenador[1] en las llamadas *Interfaces de Usuario Basadas en Modelos*.

La edición interactiva se apoya sobre un sistema de generación de páginas web dinámicas (i.e. PEGASUS), que aporta una clara separación entre contenidos y presentación, permitiendo que la información relativa a la generación de las páginas sea más accesible y analizable automáticamente por algoritmos externos. La utilización de una ontología y de una red semántica de objetos para la representación del modelo del dominio en PEGASUS, es propia de un sistema que actúa bajo el paradigma de la *Web Semántica*, otro de los campos de investigación relacionado con esta tesis. Así mismo, las constantes referencias a la web, así como a la generación dinámica de documentos HTML, hacen que el campo de la *Web Hipermedia* se haga de obligada mención en este capítulo de tesis.

[1] Del inglés: Human-Computer Interaction.

Resumiendo, en el problema abordado intervienen distintos campos entre los que podemos destacar:

a) Los sistemas de extracción automática del conocimiento en documentos web.

b) El paradigma de la Programación por Demostración.

c) Las interfaces de usuario basados en modelos.

d) Los sistemas web hipermedia.

e) El paradigma de la red semántica.

Cada una de estas áreas es un campo relevante de investigación actual en el que ha habido numerosas aportaciones a lo largo de los últimos años. PEGASUS ha sido diseñado teniendo en cuenta los apartados d) y e), mientras que DESK utiliza a), b), y c) como paradigmas principales. Del resto de herramientas se puede decir que no aportan demasiado a ningún campo de investigación en sí, puesto que fueron creadas simplemente para completar las fases de gestión y autoría del sistema expuesto en la propuesta de este trabajo. En cualquier caso, y por deferencia al resto de herramientas, podemos decir que PERSEUS manipula información contenida en el paradigma e) y que HADES ha sido diseñado teniendo en cuenta la tecnología descrita en d). Así pues, cada uno de estos campos ha sido seleccionado a partir de las herramientas y técnicas implementadas que dan lugar al trabajo de esta tesis.

Una vez aclarados estos aspectos, se procederá a realizar un repaso por la literatura relativa a los distintos paradigmas numerados anteriormente, describiendo las aportaciones más relevantes a cada uno de estos campos, materializadas en diferentes sistemas y herramientas involucradas en varios de estos paradigmas.

2.2 Extracción automática del conocimiento en documentos web

Para resolver el difícil problema de inferir la intención del usuario, DESK necesita atribuir un significado a los fragmentos que el usuario manipula. Este significado lo encuentra en el modelo del dominio a partir del cual las páginas son generadas por PEGASUS. DESK identifica partes integrantes del modelo del dominio así como de la página HTML generada por PEGASUS, localizando estructuras de la interfaz especiales, como tablas y listas, y permitiendo prestar ayuda al usuario en la transformación automática de estos elementos de la presentación.

Esta línea de actuación es posible gracias a la separación explícita, llevada a cabo por PEGASUS, entre los contenidos y la presentación. De esta forma, la aplicación de semántica a la web permite, a rasgos generales, dotar a la información contenida en la misma con la capacidad de poder ser tratada automáticamente para diversos

cometidos, permitiendo el tratar, analizar, e incluso inferir cierta información estructurada sobre los contenidos de la web, dejando ésta de ser un amasijo de datos desordenados para convertirse en una fuente de información que, mediante un tratamiento homogéneo, puede ser accedida a partir de distintos criterios, identificando rasgos comunes que permitan localizar partes constructivas a demanda de los sistemas automáticos de búsqueda y extracción.

En definitiva, se trata pues de un proceso de extracción de información, donde existen antecedentes relevantes para el problema enunciado en esta tesis que se describirán a continuación.

2.2.1 Sistemas de extracción de información.

Uno de los esfuerzos más importantes realizado bajo el paradigma de la extracción automática de conocimiento es el de los sistemas IE[2] [Mus99], los cuales permiten la extracción de datos relevantes a partir una colección de documentos. Una tarea típica de un sistema IE podría ser la de identificar, por ejemplo, los objetivos de ataques terroristas a partir de la información que aparece en los periódicos.

Una componente clave de cualquier sistema IE es lo que se denomina el conjunto de *patrones de extracción* (o reglas de extracción) que se usan para poder extraer de cada documento la información relevante. El principal inconveniente de este paradigma es que la escritura de patrones de extracción útiles es una tarea difícil que consume mucho tiempo. Es por ello que existen varios enfoques orientados al aprendizaje automático de reglas de extracción a partir de ejemplos de entrenamiento proporcionados por el usuario.

En general, los sistemas IE se pueden clasificar en tres categorías principales: los basados en restricciones sintácticas o semánticas, los basados en delimitadores y aquellos que permiten una combinación de las dos categorías anteriores. Los sistemas IE basados en restricciones sintácticas fueron los primeros en aparecer [Ril93], [Huf95], y surgen del contexto del tratamiento de texto plano, siendo su cometido el de ayudar a identificar la información relevante dentro de un documento. Por el contrario, un ejemplo típico de sistemas basados en delimitadores son los conocidos como *wrappers* [Mus99], [GRV+98], orientados a la búsqueda *on-line* de información contenida en documentos web.

2.2.2 Wrappers

Independientemente de la comunidad tradicional de sistemas IE, la generación de wrappers aparece de la necesidad de extraer e integrar datos a partir de múltiples

[2] Information Extraction.

documentos localizados en la web. Una tarea típica llevada a cabo por los wrappers es la de extraer datos de páginas web que son generados a partir de plantillas HTML predefinidas (p.e. las utilizadas en el comercio electrónico, páginas sobre climatología, o páginas de recursos turísticos). Los sistemas de inducción de wrappers generan reglas basadas en delimitadores que no utilizan restricciones lingüísticas.

En general, los wrappers consisten en una serie de algoritmos ideados para localizar información puntual en grandes cantidades de información, inicialmente bajo texto plano y actualmente bajo documentos web repartidos por la Internet. Inicialmente, los wrappers fueron algoritmos que el usuario tenía que construir manualmente a partir de *frameworks* especializados [HGN+97], posteriormente de forma semi-manual [Sod99], y actualmente las nuevas tendencias apuntan a la construcción de algoritmos de extracción de forma automática [CMM01], sin exigir del usuario conocimientos de programación para la construcción de algoritmos especializados de búsqueda en la web.

Uno de los grandes inconvenientes en la generación de wrappers es la dependencia de éstos con respecto al dominio de aplicación. Muchos de estos sistemas están basados en la detección de delimitadores (i.e. etiquetas HTML), a partir de los cuales buscan contenido explícito. Esto los hace por tanto sensibles al contexto y a los cambios de formato del documento web, siendo una directriz de obligado cumplimiento el que sean independientes del contenido y del dominio.

En algunos trabajos, como [ECJ+99], la generación de wrappers se basa en la utilización de modelos conceptuales u ontologías. La gran ventaja de estos sistemas es que existe una alta independencia de los delimitadores a la hora de extraer los datos interesantes, debido a que los modelos conceptuales describen a alto nivel los datos de interés a partir de sus relaciones, la apariencia léxica y las palabras clave, lo que los hace independientes del dominio, siendo adecuados para todo tipo de páginas web e independientes de los cambios de formato. Siguiendo un enfoque parecido existen otros sistemas, como TrIAs [BDP00] que utiliza técnicas de *Programación por Demostración* para la generación automática de wrappers a partir de un entorno demostracional que opera directamente en páginas web. En TrIAs el usuario muestra al sistema en qué información está interesado, generando éste el algoritmo de extracción adecuado a las necesidades del usuario. TrIAS genera wrappers en un lenguaje denominado HyQL, generado mediante técnicas de *Programación por Demostración* a partir de los intereses del usuario sobre información relevante seleccionada de distintas páginas web. De manera similar, DEByE [FSL+02] está basado en QBE[3] [Zlo77], un lenguaje de consulta de bases de datos para la creación de *queries* a partir de un ejemplo suministrado por el usuario utilizando un formulario, una lista de elementos o un texto seleccionado. Sistemas como TrIAS hacen uso del paradigma de los Agentes de Información [IAG], donde se automatiza el proceso de construcción de wrappers a partir

[3] Query By Example.

de intereses personalizados del usuario en la búsqueda. Este es el caso de FlipDog [FD], utilizado para la búsqueda de trabajo *on-line* en la web. En la mayoría de estas aportaciones se utilizan técnicas de aprendizaje automático, como es el caso de FlipDog donde el wrapper generado utiliza información basada en el *layaout* HTML para construir patrones de búsqueda.

2.2.3 Detección de estructuras especiales en documentos web

La mayoría de los trabajos que se han mencionado sobre sistemas de extracción del conocimiento, apuntan al reconocimiento en general de fragmentos de información a partir de unos requisitos previos de búsqueda, basados comúnmente en las preferencias del usuario. Sin embargo, existen otros trabajos orientados a utilizar estas mismas técnicas para el reconocimiento de partes o estructuras muy concretas de los documentos, como pueden ser widgets o controles HTML como tablas, listas de selección, etc.

El reconocimiento de tablas es un campo en el que ha habido varias aportaciones a lo largo de los últimos años. Las tablas son estructuras de datos muy utilizadas, debido a que son el único medio disponible para diseñar el *layout* de una página HTML. En las tablas no sólamente pueden encontrarse dependencias visuales, sino también dependencias semánticas a partir de la relación de los datos que integran dicha estructura. Este estudio ha dado lugar a distintos trabajos y tesis doctorales [Hur00] que han tratado en profundidad el problema.

En [CTT00] se propone un mecanismo para el reconocimiento de tablas en páginas HTML. El sistema está compuesto por cinco módulos que se complementan entre sí. Un módulo de análisis de hipertexto analiza el texto y extrae las etiquetas que definen la tabla en HTML. Otro módulo de filtrado de tablas utiliza reglas heurísticas para el filtrado de casos reales de tablas, pues éstas pueden utilizarse simplemente para especificar el *layout* de otros elementos de la presentación, como imágenes. El resto de los candidatos son enviados a un módulo de reconocimiento de tablas para un mayor análisis. El módulo de interpretación de tablas analiza las celdas de la tabla, estudiando la relación de los atributos y valores de cada una de las celdas. Y finalmente, el módulo de presentación de resultados se encarga de cómo representar y emplear los resultados del reconocimiento llevado a cabo.

La detección de tablas en HTML reporta interesantes ventajas con respecto a la detección en texto plano o abierto. En el primer caso se dispone de información de etiquetado sobre la disposición y geometría de la tabla, mientras que en el segundo caso se utilizan a menudo símbolos especiales para describir tablas, como tabulaciones, blancos, el punto y coma, etc. Para detectar si hay una tabla en texto plano es necesaria una desambiguación para averiguar el contenido y la disposición de cada celda de la tabla. Algunos trabajos anteriores han empleado para ello gramáticas [GK95] y métodos

de aprendizaje para el reconocimiento de tablas en este entorno [NLK99]. También, para hacer frente a este problema, tiene cabida la utilización de wrappers como método de detección automática de estructuras en documentos web. Sin embargo, una de las desventajas de los sistemas de detección wrappers basados en etiquetas HTML como delimitadores, es sin duda la poca robustez frente a cambios en la página web. Esto es debido a que cualquier cambio que involucre la modificación de las etiquetas HTML, utilizadas como referencia para localizar la información, hace que los wrappers basados en este paradigma fallen en su intento de buscar dicha información. WebCES [SBD03] intenta hacer frente a este problema mediante la caracterización de objetos en las páginas web. La idea consiste en construir un modelo basado en objetos de mayor nivel semántico sobre las distintas estructuras de la página web, donde los wrappers generados bajo este entorno trabajan abstrayéndose de los cambios realizados en las etiquetas HTML, utilizando en su lugar los objetos modelizados.

Siguiendo esta línea de investigación, uno de los últimos trabajos aparecidos es WL^2 [CHJ02], un sistema de generación de wrappers basado en aprendizaje automático que puede explotar distintas representaciones de un documento. El sistema localiza y reconoce tablas en documentos web a partir de la caracterización de dichas estructuras, creando para ello un modelo estructurado ontológico donde se representan características geométricas tales como el número de columnas, filas, número y tipo de los elementos, etc. Este sistema anota las tablas dentro del HTML para distinguir distintas características y establecer correspondencias con el modelo interno para un posterior tratamiento. Utilizando un lenguaje especial, denominado L_{tagpath}, el sistema puede generar expresiones de acceso a los datos a partir de las características modeladas. Este sistema proviene de un trabajo anterior, donde se reconocen tablas en un contexto más amplio [Hur00], a partir de texto plano bajo cualquier tipo de documento.

En otros sistemas, como en [Wan96] se sigue una modelización exhaustiva de las tablas a partir de un modelo matemático que permite operaciones de transformación en la estructura una vez modelizada. Por ejemplo, se pueden insertar, borrar, duplicar, combinar y dividir categorías de forma satisfactoria. Cada operación se define con una descripción de su efecto en el modelo abstracto construido. Este modelo permite abstraer un amplio rango de tablas, siendo independiente de la presentación de la propia tabla modelizada.

En el aspecto relativo a la edición, TABLE [BEF84] aporta un editor WYSIWYG para tablas con operadores polimórficos donde se pueden realizar operaciones comunes (borrado, inserción, etc.) de la misma forma, tanto para un carácter como para una celda de la tabla, distinguiendo en cada caso el objeto que soporta la operación. La representación básica de una tabla en este sistema se lleva a cabo mediante una matriz bidimensional. El movimiento del cursor a través de la tabla se modeliza en la matriz mediante un *cursor extendido*, que se mueve entre los objetos

lógicos registrando en todo momento información para indicar a qué nivel de granularidad están siendo editados éstos. El sistema ofrece un amplio rango de operaciones para la manipulación de tablas, construyendo un modelo que está bastante cerca de la representación física de éstas.

2.2.4 Minería de datos en la web

A parte de la identificación local de estructuras en una página web, una de las técnicas más utilizadas para la extracción de conocimiento a partir de grandes cantidades de páginas es la *minería web*. La minería web surge como resultado de la aplicación de técnicas de minería de datos al contenido, estructura, y utilización de recursos web. Esto puede ser de gran ayuda para la detección de estructuras globales y locales (modelos o patrones [HMS01]) dentro y entre páginas web. Al igual que otras aplicaciones de minería de datos, la minería web puede beneficiarse de la estructura implícita de la información (como en las tablas de una base de datos), pero también puede ser aplicada a información semi-estructurada o con poca estructuración (p.e. un formulario de libre introducción de texto). Esto implica, por tanto, que la minería web es de gran ayuda para la transformación de contenidos interpretables por los humanos a contenidos interpretables o tratables directamente por una máquina.

2.2.4.1 Retrospectiva

La minería de datos surgió al filo de los 90 [Pia91], después de varias décadas de investigación y perfeccionamiento de diversas técnicas aplicables al análisis de datos (incluyendo métodos de aprendizaje automático [HMS66], redes neuronales [MP69], algoritmos evolutivos [Hol75], y técnicas estadísticas [MS63]). Su nacimiento no se debió solamente a la solidez y madurez de dichas técnicas, sin duda alcanzada ya hacía tiempo. Además, se necesitó la conjunción de otro factor fundamental: la facilidad para disponer y compartir grandes almacenes de información [Pia00].

Debido a la reciente aparición del interés de uso de la minería de web, no es fácil encontrar antecedentes directos, y los que hay se encuentran en etapas tempranas de desarrollo. Por ejemplo, una propuesta muy reciente de una plataforma de minería web, para la mejora de la gestión del conocimiento [Ong01], deja sin determinar qué técnicas de minería de datos se incorporarán en dicha plataforma. Uno de los primeros esfuerzos por adaptar técnicas de minería de datos a la creación de una base de conocimiento a partir de páginas web es el sistema Web->KB [CDF+00], el cual convierte sus datos de entrenamiento (páginas HTML clasificadas según una taxonomía previa y relaciones entre ellas) en un conjunto de procedimientos generales capaces de reconocer a qué clase pertenecen las páginas web que el sistema visita, estableciendo nuevas relaciones entre dichas páginas.

Otros ejemplos recientes de adaptación de técnicas usadas en minería de datos para la extracción de conocimiento en web son los sistemas IndexFinder [PE00] y WebSUBDUE [MCH01]. En el primer caso, se trata de la generación automática de la página índice de un sitio web, basándose en medidas de similitud obtenidas a partir de los patrones de acceso a dichas páginas. La adaptación de las técnicas de *clustering*, previamente empleadas con éxito en tareas de aprendizaje no supervisado (con datos sin clasificar), ha dado lugar a la generación de páginas índices mejoradas. En el segundo ejemplo, los autores adaptan su propio sistema de aprendizaje basado en grafos para convertirlo en un motor de búsqueda en la web, capaz de localizar páginas no solo por su contenido, sino también porque tengan la estructura deseada de contenido e hiperenlaces. Otro ejemplo es el sistema WebOntEx [HE01], encargado de la extracción de ontologías web de forma semi-automática mediante el análisis de páginas web que están dentro de un dominio de aplicación. Las ontologías extraídas pueden ser utilizadas en distintos objetivos, como el entendimiento del contenido de la información de la web, la posibilidad de realizar distintas consultas de metadatos, el poder realizar una búsqueda más inteligente en la web y, en general, la conversión de datos HTML no estructurados en otros formatos que sí lo son, como el XML.

2.2.4.2 *Áreas de aplicación*

Comúnmente se identifican tres áreas principales de aplicación en la minería web, a saber: minería de contenidos[4], minería de estructuras[5] y minería de uso[6].

- La Minería de Contenidos es una forma de minería de texto. Para ello el recurso a tener en cuenta es en este caso una página web. La minería web de contenidos saca provecho de la naturaleza semi-estructurada de los documentos de texto de la web. Las etiquetas HTML de las páginas web, e incluso los marcados XML, albergan información que no solamente concierne al *layout* de la página en sí, sino a la estructura lógica de la propia página. De esta forma, la minería web de contenidos puede ser utilizada para detectar co-occurrencias de términos en textos (p.e. es posible que en cierto tipo de publicaciones electrónicas exista una co-ocurrencia de términos al mostrar la palabra *oro*, la cual es frecuente que aparezca junto con la palabra *plata* cuando los artículos conciernen a un dominio determinado). Otro área de aplicación es la detección de eventos (p.e. la identificación de historias en un flujo constante de noticias que corresponden a una nueva noticia o a otra previa no identificada). En este sentido es muy útil, a partir de los contenidos de una página, poder reconstruir la estructura y las relaciones existentes.

- La minería de estructura opera usualmente entre la estructura de hiperenlaces de las páginas. En este caso, el recurso a tener en cuenta en este caso es un conjunto de

[4] Del inglés: Web Content Mining.
[5] Del inglés: Web Structure Mining.

páginas web, empezando por una simple página hasta llegar a la totalidad del sitio web. Este tipo de minería de estructura web explota la información adicional contenida (a veces implícitamente) en la estructura del hipertexto. Así pues, un área de aplicación importante es la identificación de la relevancia relativa de diferentes páginas que parecen iguales cuando se analizan únicamente con respecto a su contenido.

- En la minería de uso el principal recurso a tener en cuenta en este caso es un registro de las peticiones hechas por los visitantes de un sitio web, recogidas usualmente en un log del propio servidor web. El contenido y la estructura de las páginas web, y en particular aquellas del sitio web, reflejan la interacción de aquellos que han editado y diseñado las páginas, así como la arquitectura subyacente. El comportamiento actual de quienes han usado esos recursos pueden revelar estructura o información adicional.

Como se verá en el Apartado 2.6.6, el uso de la minería de datos junto con el paradigma de la web semántica, permite dar cabida a un reciente campo de investigación conocido como el de *la minería de la web semántica*. Este paradigma permite, a grandes rasgos, la extracción del conocimiento a partir de grandes cantidades de documentos web, aprovechando la utilización de información semántica en los documentos, así como procurando la construcción de una ontología que defina los distintos aspectos y relaciones entre los datos encontrados.

2.3 Paradigma de la Programación por Demostración

DESK posee dos rasgos diferenciados. Uno de ellos es la utilización de un modelo explícito de dominio y otro es la utilización de un contexto de edición por usuarios. Esto quiere decir que la interacción del usuario es importante en cuanto a que es tomada en consideración durante todo el proceso. De hecho, la filosofía de esta tesis se enmarca dentro de lo que se ha venido a denominar, recientemente, como Desarrollo por el Usuario Final[7] [KF03], consistente en una serie de técnicas que permiten a personas que no son desarrolladores profesionales crear o modificar software. De esta forma, un aspecto fundamental en este trabajo es el de la caracterización de cambios realizados por el usuario en una página web generada dinámicamente a partir de información disponible sobre el dominio. El objetivo es ayudar al usuario a realizar modificaciones abstrayéndole de utilizar el lenguaje de creación de la página web.

En definitiva, este problema es interpretable utilizando técnicas de *Programación por Demostración*. Los sistemas llamados de *Programación por*

[6] Del inglés: Web Use Mining.
[7] Del ingles: End-User Development.

Demostración o *Programación Mediante Ejemplos* (PBD o PBE)[8] tratan de llegar tan lejos como sea posible partiendo de la facilidad de uso como prioridad principal, tomando la manipulación directa como único medio de especificación [Mye92]. Estos sistemas permiten definir aspectos avanzados de las interfaces mediante la construcción y manipulación de objetos en la pantalla, de forma que el diseñador *demuestra* el comportamiento que desea obtener sobre ejemplos concretos, y el sistema infiere la intención del diseñador y generaliza sus ejemplos a partir de la construcción de un programa que, salvo en casos muy concretos [MCM97], el usuario final no puede modificar.

El desarrollo de páginas web dinámicas se simplifica con sistemas como PEGASUS y otras herramientas basadas en lenguajes de especificación de alto nivel. Sin embargo, estos lenguajes tampoco son del todo triviales, y hay que aprender a usarlos. Lo ideal sería proporcionar al autor de aplicaciones un entorno de edición interactivo para crear los modelos. Trasladar conceptos abstractos y complejos subyacentes a estos sistemas a un entorno visual intuitivo es un problema difícil en general, y aquí es donde la Programación por Demostración puede ayudar a facilitar la tarea anteriormente comentada al usuario final.

Una de las características principales de DESK es la aportación de una interfaz WYSIWYG que permite al usuario editar páginas web sin preocuparse de cómo opera el sistema internamente. La interfaz de DESK permite al usuario abstraerse de los modelos subyacentes de PEGASUS, donde *por demostración* el usuario simplemente modifica una página web concreta generada por PEGASUS y la herramienta infiere y modifica la información procedural relativa a la generación de páginas web futuras. DESK lleva a cabo el proceso de inferencia, intentando buscar patrones de iteración para poder asistir al usuario en la ejecución automática de ciertas tareas, como la transformación entre controles HTML.

2.3.1 Entornos demostracionales.

Desde un punto de vista general, los sistemas desarrollados bajo el paradigma de la Programación por Demostración se componen básicamente de tres partes bien diferenciadas; una parte a) encargada de procesar la entrada a partir de la interacción con el usuario; b) un motor de inferencia para construir una representación programática de las acciones del usuario y c), un gestor de interacción que aplica el programa resultante para poder automatizar la ejecución en lugar de acometerla directamente el propio usuario, ya sea en el mismo instante o mediante la creación de un procedimiento que pueda invocarse posteriormente. Estos sistemas están basados en heurísticas que, dependiendo del problema a resolver, son aplicadas de una u otra forma para inferir información procedural. Así mismo ha habido intentos de aplicar técnicas de

[8] Programming By Demonstration (PBD) o Programming By Example (PBE).

Aprendizaje Automático[9], como en [LW99], y *Aprendizaje Inductivo*, en sistemas como ActionStreams [MW96], [Mau97], a este paradigma.

Atendiendo al objetivo para el que han sido diseñados cada uno de los entornos demostracionales se encuentran, entre otros sistemas, los entornos orientados a la construcción de interfaces de usuario, la creación de macros mediante la manipulación de gráficos, e incluso la creación de juegos y software educativo.

Teniendo como objetivo la construcción de interfaces de usuario, uno de los primeros sistemas basados en el paradigma de la Programación por Demostración fue Peridot [Mye88], capaz de inferir agrupamientos de componentes de la interfaz, dependencias geométricas dinámicas entre las componentes cuando se detectan relaciones de alineamiento o proporción entre tamaños, y respuestas visuales (feedback) al *input* del usuario. Peridot requiere un solo ejemplo por parte del diseñador para llevar a cabo la inferencia. Otras propuestas, como Inference Bear [FF94], [FSF95], permiten definir por demostración el comportamiento de las componentes de una interfaz. Utilizando varios ejemplos, Inference Bear identifica variables de entrada y de salida, detecta relaciones lineales entre ellas, forma sistemas de ecuaciones y utiliza las soluciones para definir las reglas de transformación de estado a aplicar como respuesta a eventos del usuario final. Su predecesor, Grizzly Bear [Fra95], amplía considerablemente el conjunto de comportamiento que puede ser demostrado con su antecesor, incluyendo comportamiento condicional así como comportamiento definido a partir del conjunto de objetos. Otra de las mejoras que incorpora Grizzly Bear es la adición de ejemplos positivos y negativos, además de los ya conocidos en Inference Bear. Grizzly Bear, al igual que otras herramientas como Pavlov [Wol97] utilizan heurísticas muy simples para inferir distintas formas de restricciones (dependencias) gráficas y comportamiento.

Otro sistema orientado a la construcción de presentaciones gráficas e interfaces es HandsOn [CS99a], [CS99b], que utiliza técnicas de Programación por Demostración que permiten al diseñador asociar valores específicos de los datos a valores concretos de la presentación, analizando las características de los datos para inferir propiedades de la visualización. El modelo de la presentación de HandsOn está basado en MASTERMIND [SSC+95] y HUMANOID [SN98], de hecho HandsOn puede verse como un editor gráfico construido sobre el sistema de presentación de MASTERMIND, en el cual el diseñador puede construir modelos de presentación mediante la manipulación directa de objetos de visualización, sin tener en cuenta cómo será representado el modelo subyacente. La principal mejora con respecto al modelo de presentación de MASTERMIND es la incorporación de ejemplos de datos de aplicación en tiempo de diseño dentro del modelo, de donde los modelos genéricos son abstraídos por el sistema. HandsOn está implementado utilizando Amulet, [MMM+97] una

[9] Del inglés: Machine Learning.

herramienta sofisticada basada en C++ que, además de proporcionar una colección de widgets, incluye un sistema de objetos que permite definir dependencias (restricciones) entre los atributos de los objetos, de forma que el valor restringido se actualiza automáticamente cuando cambia la variable independiente en tiempo de ejecución.

Por otro lado, existen sistemas demostracionales más generales que están orientados a la construcción de macros a partir de la manipulación de gráficos, como en el caso de Chimera [Cyp93], un entorno de edición gráfica que infiere dependencias geométricas a partir de múltiples instantáneas de los objetos de una escena en distintas posiciones. Chimera permite también la definición de macros (composición de acciones) generalizadas, mediante la edición y parametrización de un histórico gráfico de la secuencia de acciones del usuario. Bajo esta clasificación, Mondrian [Cyp93] es también un sistema basado en la representación gráfica de históricos, que utiliza un sistema sofisticado de heurísticas para determinar la intención del usuario, basadas en la selección de objetos relevantes (p.e. el último objeto creado), relaciones significativas entre objetos (p.e. alineamiento, proporción exacta de dimensiones) e información sobre el contexto de las acciones del usuario (p.e. puntos relevantes cercanos al ratón, tipo de objetos seleccionados).

En cuando a la construcción de juegos, una de las aportaciones más relevantes es Gamut [MM99], [McD99], [MM00], junto con su predecesor Marquise [MMK93]. En estas herramientas los usuarios no programadores pueden crear un amplio rango de software interactivo, como juegos, simuladores y software educacional. Gamut proporciona técnicas avanzadas de interacción para que un usuario poco experto pueda fácilmente expresar todos los aspectos de una aplicación. Estas técnicas incluyen el poder demostrar nuevos ejemplos mediante la selección de ciertos objetos que se consideran importantes. El sistema encuentra relaciones entre los objetos seleccionados actuales y pasados, a los cuales denomina *fantasmas temporales* de los objetos manipulados. Internamente, Gamut utiliza un árbol de decisión para representar expresiones booleanas a partir de las descripciones pertenecientes a la selección de ciertos objetos.

Otras categorías consideradas especialmente relevantes para esta tesis, incluyen la de los sistemas PBE que hacen un tratamiento de tareas iterativas mediante la monitorización del usuario, técnica utilizada por DESK para inferir información precisa sobre las acciones del usuario. También de interés son los sistemas de PBE basados en la web, que permiten servir de ayuda al usuario en la navegación mediante técnicas inteligentes y que, por otro lado, comparten algunos de los puntos relatados en este trabajo de tesis.

2.3.2 Detección de tareas iterativas

Una interesante aplicación del paradigma de la Programación por Demostración es la detección de patrones iterativos a partir de tareas repetitivas llevadas a cabo por el usuario sobre ciertos objetos, que pueden ser inferidos por el sistema y aplicados a otros objetos similares. Existen varias aportaciones a este respecto, como Familiar [PW99a], [PW99b], un agente que puede ser instruido para realizar tareas repetitivas en aplicaciones independientes del dominio, detectando ciclos de comandos y explorándolos detenidamente. Familiar utiliza un rastreador de eventos en los objetos sobre los que está actuando el usuario, así como información relativa al dominio para predecir las acciones que el usuario ejecutará en el futuro. Eager es también un agente que ayuda a automatizar tareas en el entorno multimedia HyperCard [Cyp93]. Eager detecta repeticiones y predice el próximo comando a ser ejecutado por el usuario, iluminando esta información gráficamente sobre la pantalla. Cuando el usuario decide que Eager ha aprendido lo suficiente, éste puede decidir que actúe en lugar del propio usuario.

Topaz [Mye98] es un sistema que opera bajo una filosofía similar a la descrita anteriormente, creando *Scripts por Demostración* para cualquier aplicación creada bajo el entorno de desarrollo Amulet. Topaz realiza inferencia y generaliza a partir de propuestas establecidas por defecto, explorando la estructura de datos de la interfaz y solicitando el asentimiento del usuario. Bajo esta misma filosofía, ScriptAgent [Lie99] es un sistema basado en Programación por Demostración que utiliza AppleScript para almacenar y ejecutar comandos. El usuario demuestra transformaciones realizadas en un objeto de entrada y el sistema construye un programa para transformar cualquier objeto de la misma forma. Otros sistemas, como el que se describe en [KZ97], utilizan un mecanismo similar para automatizar tareas de ejecución frecuente por medio de la monitorización del usuario para la captura de eventos de alto y bajo nivel por medio de un modelo externo de la aplicación. En este caso el sistema está semi-automatizado, y es necesaria la ayuda de un experto para analizar el histórico de eventos y mantener el modelo de la aplicación.

APE [Lie99] es un sistema que utiliza técnicas de aprendizaje automático para aprender de los hábitos del usuario de forma que, posteriormente, pueda llevar a cabo tareas repetitivas en lugar de hacerlo el propio usuario por sí mismo. El objetivo que persigue APE es, por un lado, el diseñar un agente de asistencia capaz de automatizar tareas repetitivas con un mínimo número de intervenciones por parte del usuario, utilizando técnicas de Programación por Demostración para poder llevar a cabo acciones repetitivas complejas y que interfieran lo menos posible al usuario, es decir, ofreciéndole solamente opciones adecuadas en el momento oportuno. Para lograr su cometido APE utiliza técnicas de aprendizaje automático, permitiéndole aprender rápida

y eficientemente a realizar sugerencias al usuario así como saber en qué momento debe de realizar dichas sugerencias.

En TELS [MW92] el sistema generaliza operaciones iterativas llevadas a cabo por el usuario e infiere información procedural de edición que incluye bucles y condiciones, lo que le permite llevar a cabo tareas complejas. En el caso de que la información inferida por el sistema sea incorrecta, el usuario puede corregirla de forma incremental hasta que ésta sea correcta. Siguiendo con la misma filosofía, en SMARTedit [Lie99] el sistema permite automatizar tareas de edición repetitivas mediante técnicas de aprendizaje automático. SMARTedit construye información procedural a partir de la observación de las tareas del usuario bajo el sistema, como si se tratara de la grabación de una macro. SMARTedit puede aprender a trabajar correctamente frente a nuevos casos a través del análisis del contexto en el cual se llevan a cabo las acciones del usuario. Mediante el uso de técnicas de aprendizaje automático, SMARTedit aprende información procedural a partir de unas pocas demostraciones llevadas a cabo por el usuario bajo la herramienta.

2.3.3 Asistencia al usuario durante la navegación y edición web

Una ventaja de la web como plataforma para la aplicación de técnicas de Programación por Demostración, es el hecho de que las interfaces basadas en la web son menos expresivas y hacen el problema, desde el punto de vista de la PBE, más tratable, sin que sea por otra parte la web un factor limitante en cuanto al alcance de las aplicaciones, sino todo lo contrario. Por otro lado, y en comparación con otros dominios o aplicaciones más cerradas, en la interacción con la web el acceso a los datos puede ser solucionado a partir de la detección del flujo mediante mecanismos parecidos a los que se llevan a cabo en los servidores *proxy*. En este caso la componente de Programación por Demostración puede saber la URL que el usuario ha solicitado, así como la entrada de los datos en los formularios. Otra ventaja a este respecto es la disponibilidad de los API en la mayoría de los navegadores web convencionales, es decir, de las librerías que permiten acceder a los objetos de la navegación dentro del navegador (i.e. la configuración, el documento web, los links, etc.), permitiendo construir procesos externos que capturen ciertos aspectos de la interacción con el usuario durante la navegación.

En base a la literatura existente, es posible afirmar que hay una tendencia en la que, en la mayoría de los sistemas PBD orientados a la web, la monitorización del usuario se convierte en algo indispensable como principio para analizar sus acciones y servirle de guía durante el proceso de interacción. Bajo este filosofía, en Scrapbook [SK98] los usuarios pueden demostrar en qué porciones de páginas web están interesados mediante la creación de una página personal seleccionando datos en el navegador web y copiándolos a dicha página. La interacción se lleva cabo mediante la

selección, por parte del usuario, de la información en la que está interesado, construyendo el sistema una serie de patrones basados en etiquetas HTML para la posterior detección de datos mediante técnicas de *matching* que encajen con estos patrones. Una vez que la página personal se ha creado, el sistema actualiza automáticamente los datos extrayendo los trozos seleccionados de las últimas versiones de las páginas en la web. De esta forma el usuario puede ver toda la información que le interesa en una sola página, en vez de acceder a cada una de las páginas de forma independiente. Otro trabajo relacionado de los mismos autores es SmallBrowse [Lie01], orientado a la navegación en dispositivos con una pantalla o *display* de características reducidas. La principal aportación de este navegador es que predice cuáles son los tres links más relevantes para el usuario y se los muestra como primera opción en la pantalla. La predicción de estos links se realiza mediante la inspección de un histórico con la historia de la navegación del usuario, seleccionado los tres usados más frecuentemente.

Otra de las aportaciones más relevantes en este campo ha sido Turquoise [MM97], un navegador y editor inteligente para la web donde los usuarios pueden crear páginas dinámicas por demostración en vez de hacerlo mediante la programación convencional en cualquier lenguaje de programación. Turquoise detecta y analiza la interacción del usuario, de forma que éste puede copiar fragmentos HTML en una página personal y la herramienta detecta patrones e infiere un programa en forma de *script* que servirá para generar una página personalizada a partir de la visión actualizada de la web según los datos seleccionados previamente por el usuario.

Web Macros [SCC99] es otro trabajo basado en la detección y la reutilización de tareas automatizables, cuya misión es la de emular la interacción y navegación del usuario en la red. Para ello, un conjunto de programas se encargan de observar las acciones del usuario y validarlas con los resultados finales obtenidos por el mismo. WebSheets [WSC02] es otro interesante trabajo consistente en una herramienta de autor WYSIWYG para la construcción de páginas web dinámicas que acceden y modifican bases de datos. Mediante WebSheets un usuario puede editar una página HTML y crear tablas visualmente. Una vez que ha creado la tabla en HTML el usuario puede decidir el vincular la tabla creada a una base de datos existente, configurando los distintos parámetros mediante QBE [Zlo77]. El resultado final es la inferencia, por parte de la herramienta, de una consulta en SQL que será enviada a la base de datos como resultado de las acciones llevadas a cabo por el usuario a partir de la correspondencia establecida.

Herramientas como las mencionadas en estos últimos párrafos, son de especial relevancia para esta tesis, debido a la similitud de DESK con estos sistemas. En el Capítulo V se llevará a cabo una comparación, resaltando diferencias entre estos sistemas y el propuesto en esta tesis.

Otros antecedentes relevantes son los basados en agentes de asistencia web. Estos sistemas son otra aplicación de la Programación por Demostración para la asistencia al usuario en la web que permiten, a rasgos generales, el inspeccionar las acciones del usuario como medio para guiarle por la web. Estos asistentes generalizan en gran medida las herramientas vistas anteriormente, donde también se apunta a guiar al usuario por la navegación en Internet pero, en este caso, utilizando agentes de asistencia independientes. Como ejemplo de este tipo de paradigma, LiveAgent [Kru97] promueve la idea de poder mimetizar cualquier actividad que pueda llevar a cabo un usuario en la web, pudiendo ser llevaba a cabo por los agentes como sustitutos del propio usuario. La idea es crear agentes en Java que puedan mirar la información en la web a partir de la observación de lo que el usuario hace durante su interacción en el sistema, aprendiendo de las acciones llevadas a cabo por el usuario e intentando automatizarlas posteriormente. Otro trabajo interesante a este respecto es el de los Agentes de Información [BDP+00], indicados para asistir al usuario en la localización de información relevante en vastas colecciones de documentos en la web. Esta propuesta se basa en la idea de que en muchos casos (p.e. cuando se intenta integrar piezas de información de fuentes no relacionadas) no es suficiente identificar simplemente documentos que contienen datos relevantes. En su lugar, los Agentes de Información identifican las porciones interesantes de esos documentos y los hacen disponibles para un uso futuro.

Los Agentes de Reconocimiento [LFW01] son otro trabajo que proporciona mecanismos para *mirar con anticipación* a partir de la navegación web llevada a cabo por del usuario, a través de la monitorización de la actividad del mismo, y recomendándole cuál es el mejor camino a seguir. Estos agentes infieren información sobre las preferencias e intereses del usuario a partir de la observación de sus acciones durante la interacción. Esta idea está basada en un trabajo anterior, Grammex [LN99], donde se implementan una serie de agentes que pueden ser entrenados para reconocer texto en un entorno de manipulación directa. En Grammex un usuario no experto puede definir gramáticas de forma sencilla e interactiva. Para ello el usuario presenta ejemplos concretos de texto para ser reconocidos por el agente, activando de esta forma un proceso de construcción de reglas a través de un proceso iterativo donde, a partir de una serie de heurísticas, Grammer analiza el ejemplo y visualiza un conjunto de hipótesis, emitiendo finalmente el usuario su opinión sobre las sugerencias propuestas por el sistema.

Algunas aplicaciones comerciales han incorporado de forma muy limitada ideas puntuales surgidas de la investigación en el campo de la Programación por Demostración. Sin embargo, los resultados alcanzados hasta la fecha en esta línea carecen de la fiabilidad necesaria para una amplia implantación en la tecnología de desarrollo de interfaces de usuario. El mayor problema de estas herramientas en la práctica es la dificultad para el diseñador de controlar la relación entre los ejemplos que

proporciona y el resultado que se obtiene. La inferencia acarrea inpredecibilidad y falta de control cuando el diseñador no recibe toda la información relevante sobre el estado de la aplicación y los pasos dados por el sistema.

2.4 Interfaces de usuario basadas en modelos

Basándose en la Programación por Demostración, esta tesis apunta hacia la manipulación directa y el entorno WYSIWYG como medio para asegurar al usuario final la facilidad de uso en el proceso de autoría de páginas web dinámicas. DESK funciona bajo estas premisas, donde un usuario modifica una interfaz web, es decir, un documento que sirve a su vez de interfaz para la interacción y visualización de información compartida. Mediante el entorno de edición de DESK, un usuario utiliza las capacidades proporcionadas por HTML para crear componentes de la presentación, modificando la interfaz en forma de documento hipermedia en el que el usuario puede ver directamente los cambios llevados a cabo.

En cualquier caso, el desarrollo de las interfaces de usuario es una tarea difícil y costosa. Se ha estimado que en torno al 50% del código de las aplicaciones corresponde a la interfaz de usuario, con un coste de producción también cercano a la mitad del total [Mye93], [Mye95]. Es por ello que desde principios de los ochenta existe un gran interés, tanto por parte de la industria del software como en el mundo académico, por la producción de lenguajes, herramientas y metodologías que faciliten la construcción automática de las interfaces.

Una de las propuestas más prometedoras en esta línea ha sido la que se conoce como *Interfaces de Usuario Basadas en Modelos*, que consiste en desarrollar interfaces mediante descripciones de alto nivel (modelos) de los distintos aspectos de la interfaz, en lugar de escribir el código directamente con un lenguaje de programación. En esta tesis se sigue una aproximación cercana a esta filosofía, con la utilización de modelos explícitos del conocimiento del dominio, del diseño de páginas, y de las acciones del usuario.

En el proceso de ingeniería inversa llevada a cabo por DESK, la utilización de modelos ayuda a interpretar los cambios llevados a cabo por el usuario y reconocer en una página web piezas de la base de conocimiento y bloques de presentación.

2.4.1 Desarrollo de interfaces de usuario

Bajo la filosofía de las Interfaces de Usuario Basadas en Modelos, diversos sistemas y herramientas han sido diseñados para evitar a los desarrolladores tener que utilizar un lenguaje concreto de programación empleado en la construcción de la interfaz de usuario a diseñar. En su lugar, el diseñador puede utilizar lenguajes de mayor nivel de abstracción para especificar descripciones declarativas (modelos) de las

interfaces. Entre los diferentes aspectos que se han distinguido a la hora de modelizar las interfaces, se han considerado las siguientes modelos:

- Modelo del dominio, que describe la estructura y atributos de la información que la aplicación proporciona.
- Modelo de la presentación, consistente en una descripción declarativa de la representación de la interfaz.
- Modelo de tareas, que describe las tareas que los usuarios pueden llevar a cabo bajo la interfaz.
- Modelo del diálogo, que representa una especificación de cómo será el comportamiento de la interfaz, es decir, la forma en la que el usuario final interactúa con ésta.

PEGASUS utiliza un modelo explícito para representar la información de la base de conocimiento, y también un modelo de la presentación para visualizar los objetos de la red semántica. Por tanto, son especialmente relevantes para este trabajo los antecedentes basados en la utilización de modelos del dominio y de la presentación. También son relevantes los trabajos que hacen uso de un modelado de tareas. Aunque DESK no sigue un modelado clásico de tareas tal y como se hace en otros antecedentes, se utiliza sin embargo cierta información de alto nivel sobre las acciones del usuario, con objetivo de proporcionar a éste ayuda en la edición de una página web. Respecto al modelo del diálogo, como ya se dijo en el Capítulo I, en esta tesis se prestará mayor atención a lo relacionado con presentación y no con el comportamiento de la interfaz.

El papel del modelo del dominio varía significativamente entre los distintos antecedentes, reflejando a ese respecto diferencias sustanciales entre unos sistemas y otros. Por ejemplo, en ADEPT [JWJ94], [WJ96] el modelo del dominio es minimalista, y está compuesto exclusivamente por una lista de tipos de entidad. Las asociaciones entre el modelo del dominio y otros modelos son representados a un nivel alto de abstracción. Esto es debido a que ADEPT está especialmente enfocado a la identificación de los requerimientos del usuario, y no a la generación completa de interfaces de usuario funcionales. Por otro lado, UIDE [BFM92] utiliza un modelo extendido de datos para generar el *layout* de la interfaz de usuario. De forma parecida, Mecano [Pue96] y MOBI-D [PE99] utilizan un modelo del dominio a partir del cual se genera el *layout* y el comportamiento dinámico de la interfaz. En estas herramientas es posible definir modelos del dominio mediante la utilización de un lenguaje de *frames*, que define jerarquías de clases, las cuales puede tener un número distintos de propiedades asignadas denominadas *facets*. Como mejora sustancial sobre Mecano, MOBI-D incorpora además un modelo de tareas jerárquico.

A la expresividad de los modelos, Mecano añade la utilización de formularios para editar y crear los distintos elementos del modelado. Mecano [Pue96], y más en concreto su sucesor MOBI-D [Pue97], [PE99], [EP98] como entorno de construcción de interfaces de usuario, incluye varias herramientas para ayudar al diseñador en sus tareas [Mob]. Por un lado U-Tel, utilizada por expertos del dominio, permite la creación de modelos informales de tareas de usuario. TIMM es un analista basado en conocimiento que permite el diseño de la presentación y el diálogo. MOBILE se utiliza como herramienta para la construcción de la interfaz y editor interactivo de tareas. Finalmente Model Editors es un conjunto de herramientas interactivas para la definición e inspección del modelo de la interfaz.

El modelo de la presentación es utilizado de forma explícita por algunas herramientas de generación de interfaces de usuario. Este es el caso de MASTERMIND [SMF+94], [CSS97], que permite la definición de un potente modelo de la presentación capaz de adaptar la información de la interfaz a distintas plataformas de acceso. El modelo de la presentación de MASTERMIND está basado, en su mayoría, en ideas extraídas de otras herramientas, como HUMANOID [SLN93], [SN98] e ITS [WBB+90], [WB90]. HUMANOID define un elaborado modelo de la interfaz que incluye componentes de la aplicación, la presentación y el diálogo. Los desarrolladores construyen modelos de aplicación, y HUMANOID escoge entre un conjunto de plantillas para visualizar la interfaz. El desarrollador puede refinar el comportamiento de la interfaz mediante la edición del modelo del diálogo. HUMANOID asiste al diseñador, aunque no de forma automática, en la generación de la especificación del comportamiento dinámico de la interfaz, y requiere un esfuerzo considerable de éste para generar interfaces que no concuerden con sus plantillas predefinidas, como suele suceder a la hora de diseñar interfaces con cierto grado de complejidad.

En algunas de las herramientas de generación de interfaces basadas en modelos, el modelo de la presentación es simplemente un conjunto de widgets proporcionados por la toolkit utilizada en la implementación; algunos sistemas ofrecen al diseñador asistencia en seleccionar el widget apropiado. Este es el caso de MASTERMIND, que permiten la personalización de prototipos de widget. Otros sistemas como TRIDENT [VB93], [BHL+94], [BHL+95] poseen un modelo abstracto y concreto de la presentación. En TRIDENT las entidades en el modelo abstracto representan un paso de refinamiento de la interfaz, permitiendo a los desarrolladores el posponer decisiones sobre el widget a ser utilizado hasta después del proceso de diseño.

HUMANOID también permite la creación de los modelos mediante editores gráficos especializados. HUMANOID incorpora una interfaz de diseño fácil de utilizar, que permite a los diseñadores crear complejas interfaces sin la necesidad de emplear ningún lenguaje de programación.

ITS se compone de una arquitectura de cuatro capas, cada una con sus correspondientes herramientas para la computación *back-end*, control del diálogo, la creación de interfaces basados en reglas y la implementación de primitivas gráficas utilizadas en el estilo de la presentación. La filosofía de ITS está basada principalmente en la separación de la aplicación y la interfaz de usuario, a partir de una clara división entre distintos aspectos relativos a la composición de una interfaz, teniendo en cuenta la diversidad de usuarios y configuraciones a las que el diseño debe de hacer frente de forma universal. ITS y MASTERMID exigen de los desarrolladores el especificar explícitamente todos los niveles de modelado, estos sistemas no ofrecen herramientas de diseño automático aunque, por el contrario, permiten la capacidad de reutilizar especificaciones.

Una de las últimas apuestas relacionadas con la construcción de interfaces de usuarios es el lenguaje XIML [PE02], el cual permite una representación común entre *datos de interacción*, es decir, los datos que definen y relacionan todos los elementos relevantes de una interfaz de usuario. XIML soporta el diseño, operación, organización y funciones de evaluación, además permite relacionar elementos concretos y abstractos de una interfaz, permitiendo a los sistemas basados en conocimiento el trabajar con los datos capturados. A rasgos generales, XIML es una colección de elementos de la interfaz categorizados en uno o más componentes (tareas, dominio, usuario, diálogo y presentación). En este sentido el lenguaje no está delimitado en cuanto al número y tipo de los componentes que pueden ser definidos. El principal uso de XIML es el diseño de interfaces de usuario independientes de la plataforma de acceso, haciendo esto posible gracias a una estricta separación entre la definición de la interfaz y la renderización de la misma. De esta forma XIML permite la conversión directa a lenguajes tan populares como HTML y WML, siendo éste un marco ideal para la ingeniería inversa de páginas de HTML a XIML y viceversa. En este sentido trabajos como VAQUISTA [VBS01] permiten procesar el código HTML para componer en XIML un modelo explícito de la presentación, a partir de la aplicación de reglas de correspondencia entre los distintos componentes que forman el HTML y los elementos de presentación XIML, permitiendo así la ingeniería inversa de documentos web. Una versión posterior permite además la re-ingeniería de páginas web utilizando también un modelo del diálogo en XIML [BVE02]. XIML está basado en trabajos anteriores, como MOBI-D, a partir de la utilización de características comunes referidas a la modelización de interfaces de usuario.

2.4.2 Modelización de tareas

Una importante contribución surgida en el ámbito de las interfaces basadas en modelos [Sze96] es la que propone el análisis de tareas del usuario como elemento fundamental en la modelización, desarrollo y explotación de las propias interfaces. La representación basada en tareas enfatiza la importancia de la comprensión de las tareas

del trabajo de los usuarios por parte de los diseñadores, y las consecuencias que los cambios en el diseño implican en las tareas. Esto sitúa a las personas y sus tareas en el punto de partida del proceso de diseño, de forma que el desarrollo de las interfaces no es un mero proceso iterativo de ensayo y error, sino un proceso informado desde el principio sobre las tareas que el sistema ha de soportar. Las ideas de diseño están motivadas desde la perspectiva de las tareas del usuario.

DESK utiliza un modelo de acciones del usuario (i.e. modelo de monitorización) que podría considerarse, en general, como un modelo de tareas de bajo nivel de abstracción creado en tiempo de ejecución. Éste es usado por las distintas heurísticas internas para obtener el máximo de información posible a la hora de realizar inferencia. En general DESK no utiliza un modelo abstracto de tareas como el que utilizan algunos de los antecedentes aquí descritos. DESK únicamente cataloga las acciones para asignarles información semántica, acciones por otro lado relacionadas con la edición del documento web por parte del usuario.

El modelado de tareas describe las operaciones que el usuario final puede realizar con una aplicación. Típicamente esta descripción se hace en forma de descomposición jerárquica de cada tarea en los pasos (subtareas) necesarios para llevarla a cabo.

Uno de los trabajos más notables en la modelización de tareas es ConCurTaskTrees [Pat01], que aporta una especificación del modelado de tareas utilizado para diseñar aplicaciones interactivas. Su cometido es proporcionar una notación fácil de utilizar que pueda soportar el diseño real de aplicaciones industriales. ConCurTaskTrees se basa en una estructura jerárquica donde el problema a modelizar se descompone en problemas más pequeños, lo cual proporciona un alto grado de granularidad, permitiendo la reutilización de los diseños independientemente de su tamaño, así como la definición de los mismos a distintos niveles semánticos. ConCurTaskTress diferencia entre cuatro tipos de tareas: tareas de interacción, de aplicación, de usuario y tareas abstractas. La flexibilidad y riqueza expresiva de la notación para el modelado de tareas utilizada en ConCurTaskTrees, permite representar concurrencia y actividades interactivas, así como la posibilidad de soportar interrupciones y cooperación entre múltiples usuarios. Un aspecto clave en esta notación es la habilidad de suministrar gran cantidad de información de forma intuitiva, sin abusar de una expresividad excesiva de cara a los diseñadores. Esta característica de ConCurTaskTrees ha sido contrastada mediante la utilización de esta notación en el mundo empresarial por personas sin conocimientos específicos en informática.

MOBI-D también utiliza un modelo jerárquico para la representación de las tareas del usuario bajo la interfaz. El modelo descompone las tareas en subtareas, en forma de estructura de árbol, permitiendo la definición de atributos e información procedural (p.e. cuándo una cierta subtarea debe ser ejecutada en secuencia). Las

condiciones que afectan a la ejecución de una tarea/subtarea pueden ser especificadas directamente en el modelo. En MOBI-D, los modelos de tareas son elicitados por expertos del dominio y pueden ser definidos mediante desarrolladores de la interfaz. La forma de interactuar con MOBI-D se lleva a cabo a partir de una especificación semi-informal, proporcionada por el experto, basada en una categorización de objetos, acciones y actores implicados en las tares que el usuario necesita realizar. A partir de esta especificación el sistema genera un esqueleto del modelo de tareas. El diseñador refina el modelo inicial generado hasta obtener una especificación completa de las tareas, a partir de la cual el sistema genera automáticamente la interfaz teniendo en cuenta preferencias y estrategias que el diseñador puede indicar a distintos niveles de detalle.

MASTERMIND [SSC+95] utiliza un modelo de tareas en el que éstas son clasificadas de mayor a menor nivel de abstracción, donde las tareas más abstractas son anotadas con conceptos del dominio. Las tareas de menor nivel de abstracción, localizadas en los niveles inferiores del árbol, se corresponden con relaciones directas sobre la interfaz, siendo, finalmente, las tareas terminales (i.e. hojas del árbol) las vinculadas directamente con acciones elementales o eventos del usuario durante la interacción con la herramienta (p.e. pulsar un botón, seleccionar un elemento de un combo box, etc.). Estas tareas están asociadas a componentes gráficas de la interfaz, donde se incluye información sobre la respuesta interna del sistema, así como la respuesta visual de las componentes a las acciones del usuario.

ADEPT lleva a cabo el modelado de tareas a partir de un modelo abstracto de las componentes gráficas de la interfaz, automatizando el sistema la selección de componentes concretas a partir de las abstractas. De manera similar TRIDENT utiliza, además de la descripción de tareas, un modelo explícito de los datos que representa propiedades como el tipo, estructura, dominio de valores o precisión de los datos. ADEPT y TRIDENT automatizan gran parte de la construcción de interfaces a cambio de una pérdida de generalidad en cuanto al tipo de interfaces generadas.

La información que representan los modelos de tareas permite recoger la progresión entre los distintos niveles de abstracción en las operaciones llevadas a cabo por el usuario bajo la interfaz. Esta información es útil para el desarrollo de sistemas de ayuda, seguimiento y asistencia al usuario.

Una dificultad inherente de cara a los usuarios que manipulan una interfaz es cómo utilizar las capacidades de la aplicación para realizar las tareas deseadas. El grado de libertad que permiten las interfaces de usuario tiene como consecuencia una mayor complejidad en el manejo de éstas. Las aplicaciones tienen diferentes tipos de comandos y opciones, de modo que no es fácil para el usuario memorizar y recordar la sucesión de comandos necesaria para realizar una tarea. Los sistemas de ayuda proporcionan a los usuarios indicaciones sobre cómo pueden llevar a cabo sus tareas. Sin embargo, es

posible que al usuario no le interese recibir indicaciones demasiado detalladas sino que simplemente desee llevar a cabo su trabajo de la manera más rápida posible.

Muchas aplicaciones tan populares como Microsoft Office y otras competidoras intentan responder a este problema ofreciendo asistentes [HBH+98] de tareas o *wizards*. El comportamiento típico de un asistente es analizar el contexto actual, plantear al usuario una secuencia de preguntas sobre cómo desea que se realice la operación, y finalmente realizar la operación. Estos asistentes hacen las tareas mucho más fáciles porque llevan al usuario a través de un árbol simple de alternativas limitadas, en lugar de exponerle a un nivel de libertad mucho mayor en el modo normal de operación.

DESK incorpora un agente personalizado de asistencia que analiza las acciones del usuario para prestarle ayuda en la manipulación de estructuras complejas de la interfaz, como transformaciones automáticas entre widgets. Aunque el agente de inferencia DESK no se apoya en un modelo formal de tareas, éste analiza las acciones de bajo nivel llevadas a cabo por el usuario en la edición del documento web, obteniendo así información de alto nivel sobre lo que el usuario pretende hacer en la interfaz de edición. De esta forma, mediante la monitorización y el análisis exhaustivo de las acciones, es posible proporcionar al usuario final un entorno de asistencia que permita reducir en gran medida la complejidad de ciertas operaciones, así como automatizar tareas pesadas de llevar a cabo.

TWIW [Con98] utiliza un modelo de tareas para llevar a cabo un seguimiento del usuario durante la interacción con el sistema, analizando las tareas que se corresponden con cada acción acometida. Para lleva a cabo su objetivo TWIW utiliza un monitor de tareas, captando los eventos producidos por el usuario e intentando inferir la intención de éste para completar tareas iniciadas, mostrándole además los pasos a seguir en cada caso. La idea es proporcionar un sistema de ayuda y guiado del usuario en la interacción con la interfaz.

2.5 Sistemas web hipermedia

La utilización y diseño de interfaces de usuario ha sufrido en los últimos años un considerable cambio desde la aparición de la tela de araña o web, utilizada comúnmente para navegar por la información disponible en distintos servidores de la Internet. La creación de una interfaz de acceso único, a través de un navegador, permite que todos los usuarios puedan compartir la información sin los problemas subyacentes de la diversidad de la plataforma de acceso, adoptando HTML como lenguaje de representación de los datos, y evitando de esta forma los inconvenientes de la utilización de distintos lenguajes de especificación para la presentación de la interfaz.

Es conveniente poner de manifiesto que, en sus comienzos, HTML no permitía especificar información procedural, sino solamente datos con estilo destinados a la

presentación de la información a partir de ciertas etiquetas pertenecientes al propio lenguaje, las cuales son renderizadas utilizando un modelo de vista estándar para la mayoría de los navegadores del mercado. Posteriormente se incorporó información procedural añadiendo nuevas especificaciones al lenguaje y mediante la utilización de sublenguajes embebidos, como por ejemplo JavaScript, o la invocación de programas como Applets o CGIs. Hoy en día la compatibilidad entre los aspectos declarativos de la presentación de los datos y el aspecto procedural se ve facilitada con lenguajes como JSP o ASP, que permiten además embeber lenguajes de programación de alto nivel como por ejemplo Java. Toda esta tecnología permitió generar documentos de forma dinámica, y como caso particular construir páginas web generadas dinámicamente.

A este respecto, DESK permite al usuario modificar una interfaz web, en forma de documento hipermedia. La página HTML juega el papel de interfaz de usuario, integrando elementos de la presentación propios (i.e. controles HTML) para la interacción con el usuario, así como conocimiento compartido de forma universal. Aprovechando además la separación explícita que hace PEGASUS de presentación y contenidos como sistema hipermedia adaptativo, es posible editar, mediante DESK, los aspectos referentes a la interfaz o documento web generado dinámicamente.

2.5.1 Retrospectiva

El origen de los sistemas hipermedia [Nel87] se sitúa en distintas aplicaciones utilizadas durante décadas para el manejo de lo que se denomina el hipertexto (ver por ejemplo [Nei89] y [Hyp89]). Hipermedia es el término que define el almacenamiento y recuperación de información mediante un ordenador de una manera no secuencial. Como extensión del término hipertexto (escritura no secuencial), hipermedia implica enlaces (links) y navegación en un material almacenado en cualquier medio, léase texto, vídeo, sonido, música, gráficos, etc., siendo la red el medio más habitual en nuestros días para la utilización del hipertexto, cuya utilización temprana es patente [CA99].

La utilización de la hipermedia [ORB+95] se justifica mediante el incremento de la tecnología actual y los niveles de requerimiento para la manipulación y compartición de datos. Es obvio que la capacidad de los dispositivos de almacenamiento empieza a ser considerable, y los métodos tradicionales de búsqueda decaen en su uso por esta razón. La utilización de multimedia, que mezcla texto, figuras, datos, sonido o vídeo, hace más ricos los documentos a manejar. La utilización del hipermedia redunda en grandes beneficios cuando se trata de manejar grandes bases de datos de distintos medios, como por ejemplo enciclopedias, diccionarios o libros de referencia que son leídos precisamente de forma no secuencial.

2.5.2 Sistemas web hipermedia adaptativos

La principal misión de los sistemas web hipermedia es la de facilitar a los usuarios el navegar a través de la información que más les interese. Sin embargo puede ser necesario añadir una adaptación automatizada, donde el sistema se adapte a las necesidades de los usuarios [Enc97b]. De esta forma surgen los sistemas adaptativos hipermedia dentro de los sistemas de generación dinámica hipermedia, cuya principal misión es la de adaptar la información presentada a cada usuario en particular [Car01]. En estos sistemas la información no permanece estática en páginas web creadas previamente, sino que es generada dinámicamente para poder ver dicha información de diferentes formas en cada momento. Desde un punto de vista estricto, un sistema adaptativo hipermedia es aquel que es capaz de cambiar el contenido y la presentación de los distintos nodos de información, así como alterar las estructuras de enlaces o anotar links, a partir de un modelo de usuario existente [Bra99]. Esta funcionalidad puede ser llevada a cabo mediante la aplicación de tecnología web estándar, como el uso de CG-Scripts; Java Servlets, Active Server Pages, etc.

Para adaptar la información presentada a cada usuario es necesario registrar en cada momento información de las aciones del usuario, así como deducir cómo evoluciona el usuario durante la interacción. Basada en esta abstracción del estado del usuario, el sistema puede decidir cómo llevar a cabo la adpatación. La representación del estado del usuario es lo que se denomina el modelo del usuario, el cual contiene aspectos que son indicados explícitamente por el propio usuario, como el color o preferencias del medio, el tipo de aprendizaje, el conocimiento, y otras características que pueden ser seleccionadas a partir de un cuestionario. No obstante, la parte más interesante de un modelo del usuario es la información que el sistema mantiene sobre la relación del usuario con respecto a los conceptos del dominio, obteniendo esta información a partir de la observación de comportamiento del usuario durante la interacción [Küh93].

2.5.2.1 Clasificación

Se puede establecer una clasificación de los sistemas adaptativos hipermedia [BKV98] a partir de a) el área de aplicación, b) las características del usuario a las que adapta, c) los aspectos que pueden ser adaptados y d) los métodos y técnicas de adaptación.

a) El área de aplicación de los sistemas adaptativos hipermedia comprende diferentes dominios, donde podemos encontrar distintos tipos de sistemas adaptativos, los más importantes son:

- Sistemas educativos, donde el material didáctico es ofrecido al usuario, en este caso al estudiante, adaptándolo de acuerdo a sus necesidades y características de aprendizaje, existiendo distintos paradigmas para llevar a cabo la adaptación [TM98]. En este tipo de sistemas el hiperespacio está relativamente acotado y contiene los materiales relacionados con los conceptos contenidos en el curso.

- Sistemas de comercio electrónico o e-commerce, los cuales utilizan la adaptación del contenido, en este caso los productos a vender, en función de las posibilidades o de los gustos del usuario. Este tipo de sistemas ha proliferado mucho en los últimos años, siendo posible encontrarlos en gran número de empresas o sitios de venta *on-line*.

- Sistemas de información y ayuda *on-line*, que tienen por cometido suministrar información o ayuda a los usuarios con distintos niveles de conocimiento sobre un tema en cuestión. Cada nodo del hiperespacio representa un concepto y contiene varias páginas de información. Los usuarios pueden tener distintos objetivos, conocimientos previos y preferencias.

- Sistemas de información institucional y gestión de vistas personalizadas, donde se le muestra al usuario información parcial o total a partir de su categoría laboral o función dentro del organismo en cuestión.

b) Las características del usuario que son utilizadas para asegurar una adaptación correcta de la información en cada caso. De esta forma es esencial identificar qué características pueden variar de unos usuarios a otros e incluso, para un mismo usuario, de un momento a otro. Generalmente, estas características están relacionadas con su contexto de trabajo y con algunos aspectos personales, como el conocimiento, los objetivos, la procedencia, la experiencia en la navegación y las preferencias.

c) Los aspectos que pueden ser adaptados como, básicamente, los contenidos y los enlaces. En el primer caso se trata de adaptar el conocimiento del dominio utilizado para presentar los datos al usuario. La mayor parte de los trabajos realizados a este respecto provienen de la adaptación de textos a los usuarios [Bru92], [Bru98]. En cuanto a la adaptación de los enlaces, u opciones de navegación, se trata de ofrecer ayuda a los usuarios de forma que puedan encontrar su camino en el hiperespacio adaptando la forma de presentar los enlaces a los objetivos, conocimientos y otras características de los usuarios.

d) Las técnicas de adaptación, como procedimientos para permitir que el sistema se adapte a los usuarios que acceden al mismo, y los métodos de

adaptación definen una generalización de una técnica de adaptación existente. Un método puede ser implementado mediante diferentes técnicas. Al mismo tiempo, cada técnica se utiliza para implementar distintos métodos utilizados en la misma representación del conocimiento.

2.5.2.2 *Ejemplos de sistemas hipermedia adaptativos*

Las aportaciones más relevantes al campo de la hipermedia adaptativa vienen de la mano de los sistemas *Courseware* [ROB95] hipermedia, o sistemas educativos hipermedia adaptativos. Estos sistemas son relevantes para el trabajo de esta tesis en cuando a que son sistemas que se han preocupado de cómo representar el conocimiento del dominio, al igual que ocurre en PEGASUS. Entre esos sistemas podemos mencionar:

- HyperTutor [PGL95] almacena datos sobre las características del estudiante con respecto al tema objeto del estudio (novato, medio, experto), conocimiento del dominio (conceptos estudiados), ejercicios, así como un historial del aprendizaje. El sistema está basado principalmente en dos componentes: el componente hipermedia y el componente tutor.

- ELM-ART [BSW96] y ELM-ART II [WS97], desarrollados por el grupo ELM consisten en un curso de enseñanza de LISP basado en ELM-PE, un entorno inteligente de enseñanza. En ELM-ART el material educativo se compone de páginas web que están asociadas a los conceptos que deben de ser aprendidos, relacionados entre sí mediante relaciones que indican qué conceptos son requisitos previos de otros y cuales se suponen aprendidos. La interacción con el usuario queda reflejada en un modelo del estudiante, y al presentar la información a los estudiantes los enlaces son anotados siguiendo la metáfora del semáforo, es decir, colocando un indicador rojo a la izquierda de un enlace que refleje que el estudiante no está preparado para abordar el concepto, o un indicador verde que refleje que el estudiante sí está preparado para abordar el concepto. Un indicador amarillo advierte que el estudiante está preparado para abordar el concepto, pero el sistema por el contrario no recomienda hacerlo.

- Interbook [BES98] está basado en la arquitectura de ELM-ART, describiendo la aplicación del dominio mediante un documento estructurado jerárquicamente en forma de libro electrónico. Las secciones del documento son asociadas con conceptos del dominio, de esta forma mediante un procedimiento especial se convierte el documento en un conjunto de páginas HTML, asociando los conceptos a dichas páginas. Cada página tiene prerequisitos sobre el conocimiento, además de unos conceptos de salida que se generan a partir del documento hipermedia. Interbook anota los enlaces con distintos objetos gráficos y estilos para dirigir la

navegación del alumno por los distintos objetivos del curso, en función de su perfil o modelo de usuario.

- AHM [SVD+98] ofrece a los usuarios una guía, tanto local como global, mediante la utilización de las técnicas de ocultación de enlaces y anotaciones, incluyendo también mapas conceptuales en el modelo. La estructura del hiperespacio se presenta mediante un conjunto de nodos y enlaces. Los nodos pueden representar conceptos o documentos, y los enlaces representan las relaciones entre los nodos. Cada enlace tiene asociado un peso que representa, en el caso de la relación entre dos conceptos, la cantidad de conocimiento sobre un concepto que debe poseer un estudiante para acceder a otro concepto, y en el caso de relaciones entre un concepto y sus contenidos, la dificultad del documento con respecto al concepto que explica.

- AHA [BC98] es una herramienta que tiene como características principal el poder adaptar la información mediante la inclusión de condiciones en el propio código HTML, actuando dichas condiciones como filtros a la hora de decidir qué información debe de mostrarse al estudiante y cuál no. AHA mantiene un modelo del estudiante utilizando valores lógicos para indicar si el estudiante conoce o no un concepto determinado. La ventaja principal del sistema es su sencillez, dado que está basado únicamente en páginas HTML con comentarios adicionales que codifican las condiciones sobre la visualización de un fragmento HTML. Por el contrario, una de las mayores desventajas es el mantenimiento en el sentido en el que no existe una clara separación entre los contenidos y la secuencia del curso, esto hace que a medida que crece el curso su mantenimiento se convierta en una tarea complicada.

- C-Book [KK94] es un sistema similar a AHA, donde igualmente los contenidos del curso se estructuran en distintas páginas HTML, las cuales incluyen instrucciones condicionales en el lenguaje de programación C++ que serán evaluadas para llevar a cabo la adaptación sobre los contenidos.

- DCG [Vas97] permite la generación automática de cursos que se adaptan a los objetivos de los estudiantes, a su conocimiento previo del dominio y al proceso de aprendizaje. En este sistema se separa la estructura conceptual del dominio de los contenidos del curso, estructurándose los conceptos como un mapa de carreteras que se utiliza para generar el plan del curso. Para ello, un planificador busca aquellos subgrafos que conecten los conceptos ya conocidos por el estudiante con el concepto objetivo del curso, y le ofrece al estudiante un plan. El proceso de diseño de un curso en DCG consiste en crear la estructura de conceptos y añadir enlaces que relacionen cada concepto con los ficheros HTML disponibles.

- APHID [KTG00] es un prototipo que genera semi-automáticamente aplicaciones hipermedia personalizadas basadas en un conjunto de elementos hipermedia, un

modelo del dominio y un modelo del estudiante. APHID crear un mapa de conceptos que almacena las asociaciones entre dichos conceptos y clasifica los datos en función del tipo de instrucción que representan (test, explicación, simulación, etc.).

- TANGOW [Car01] es una herramienta de generación dinámica de cursos basada en un modelo de tareas y reglas docentes. Las tareas se corresponden con unidades básicas de conocimiento en las que se establecen referencias a contenidos a partir de páginas HTML. Estas se distribuyen a partir de una subdivisión entre tareas y subtareas, llevada a cabo mediante las reglas de aprendizaje. El cometido final de esta modelización es la de presentar al alumno ciertos conocimiento en función de su progresión durante el curso, guiando a este por los distintos conceptos u objetivos de aprendizaje. Las tareas en TANGOW pueden ser de tres tipos distintos: teóricas, prácticas y ejemplos, presentando al alumno unas u otras tareas en función de la evaluación en tiempo real de las reglas que subdividen dichas tareas.

Otro campo que cuenta con gran aceptación para la utilización de la hipermedia adaptativa es del comercio electrónico, donde se pueden encontrar aportaciones provenientes del mundo académico, como TELLIN [Joe99] que genera documentos individuales en tiempo de ejecución personalizados para cada tipo de usuario, teniendo en cuenta sus intereses personales a la hora de comprar. SETA [ABG99] es otro trabajo que presenta una arquitectura para la construcción de tiendas web adaptativas para personalizar la interacción con el usuario, ofreciendo a éste los productos que mejor se adecúan a sus necesidades y adaptando la descripción del catálogo de productos a sus preferencias y experiencia previa. A parte de los sistemas vistos anteriormente, otros trabajos, como el navegador AVANTI [PSS01] que utiliza adaptabilidad basada en la información disponible a partir de conocimiento de la interacción con el usuario. El objetivo principal de AVANTI es el de facilitar la interacción a individuos con distintas habilidades, conocimientos, requerimientos y preferencias, incluyendo a personas con algún tipo de discapacidad o incluso personas de edad avanzada. AVANTI está basado en la Metodología Unificada de Desarrollo de Interfaces de Usuario (U2ID), ideada por los mismos autores.

2.5.3 La autoría en los sistemas web hipermedia

La información contenida en los sitios web es susceptible de ser modificada cada cierto tiempo. En concreto, en ciertos tipos de sistemas la información debe ser actualizada con una frecuencia mucho mayor (periódicos, revistas, tablones de anuncios, etc.). En general, la modificación de documentos web contenidos en estos sistemas no es una tarea obvia, pues obliga al usuario a aprender correctamente HTML. Los editores de HTML incluidos hoy en día en los navegadores más convencionales simplifican enormemente esta tarea.

Sin embargo, existe una dificultad en los sistemas hipermedia que generan información dinámica, como es el caso específico de los sistemas hipermedia adaptativos donde la información es generada dinámicamente a partir de un modelo del usuario y/o de la plataforma de acceso. En este tipo de sistemas no es posible realizar una edición de la página utilizando herramientas de edición estáticas convencionales. Surge aquí un problema que no ha sido aún solucionado de una manera convencional, a diferencia de la edición de páginas web estáticas, que además es dependiente de la forma en que se generan las páginas y de la aplicación web encargada de la adaptación en sí.

El problema de la edición surge reiteradamente en varios de los paradigmas abordados hasta el momento en la tesis. En general, es posible afirmar que la utilización de una especificación para la generación de una interfaz (como es el caso de los documentos web), reporta una dificultad inherente para el usuario final en la manipulación de este lenguaje, siempre que éste no sea un experto en el lenguaje de especificación en sí. Por tanto, y atendiendo en general a los sistemas hipermedia dinámicos, la edición se convierte en uno de los aspectos más oscuros, donde sólamente se han encontrado algunas soluciones parciales al problema. La inexistencia de una estructura detrás de los documentos convierte el mantenimiento de los sistemas hipermedia dinámicos en una tarea complicada [Car01].

Respecto a la edición en los sistemas hipermedia adaptativos, como ejemplo de sistemas hipermedia de generación dinámica de documentos, uno de los aspectos susceptibles de ser editado o gestionado con relativa sencillez es el de los contenidos [NO01], [BBG+01], sobre todo en sistemas donde éstos están separados completamente de la estructura de la navegación, como en Arthur [GH99], o ELO [SAD01] basado en el paradigma de los *Objetos de Aprendizaje*. En este caso los enlaces son externos y fácilmente actualizables. En los casos en los que no haya separación explícita entre los contenidos y otro tipo de información acerca de la navegación, como en AHA[BC98], esta tarea no es nada sencilla, ya que para llevar a cabo modificaciones el diseñador debe sumergirse en la aplicación y realizar los cambios oportunos. Algunos sistemas han tratado de solventar este problema mediante la utilización de bases de datos, donde se almacena tanto información conceptual como procedural sobre la navegación (como en AHM [SVD+98], Arthur y TANGOW [CPR99], [Car01]), aunque otros aspectos siguen siendo difíciles de cambiar, como la presentación y su relación con los contenidos.

Algunas herramientas de autor, como ATLAS [MC00], [MC01d] para TANGOW permite la construcción de cursos mediante una interfaz de usuario basada en la manipulación directa de objetos, de esta forma se evita la necesidad de tener que utilizar directamente un lenguaje de modelado específico. Mediante ATLAS un usuario puede diseñar la estructura de un curso TANGOW, asignando propiedades y contenidos, y pudiendo simular la generación completa del curso a partir de un perfil

definido del usuario. Sin embargo, nuevamente el usuario debe de hacerse con el lenguaje visual abstracto de la herramienta, así como con el formalismo utilizado por TANGOW para representar los cursos, en este caso mediante tareas y reglas docentes.

Otros sistemas orientados a la creación de cursos adaptativos, intentan hacer frente al problema de la autoría mediante la edición parcial de los componentes o la edición total frente a requerimientos muy limitados y ligados, comúnmente, a la propia explotación del curso bajo el sistema. Bajo esta filosofía, ECSAIWeb [SG00], un sistema basado en la web, permite la creación de sistemas de enseñanza adaptativos mediante un proceso de desarrollo en cinco pasos: definición de los puntos de evaluación, determinación de los objetivos pedagógicos, descripción de las unidades de conocimiento, edición de las unidades de conocimiento físicas, y la validación de la coherencia de todos los componentes. MacroNodeEditor [PBP00] aporta, en este sentido, diversas herramientas para presentar el formalismo de definición de cursos a los diseñadores con el fin de describir plantillas y anotaciones sobre los datos. Esta herramienta se basa en el concepto de *macronodo* que es, básicamente, un conjunto de información que puede ser presentada de distintas formas dependiendo de distintos tipos de contextos. Otros sistemas utilizan SGML para la estructura de los documentos, como en [GBP98], donde se usa una presentación fija y que no varía para cada estudiante, o en [SBF+00] donde esta estructura se representa directamente en XML.

En general, el diseño de herramientas de autor eficientes no es un problema que sea exclusivo de los sistemas web hipermedia. Desde siempre en otros ámbitos más generales, como en la construcción de software educativo, ha existido este problema. En este sentido Eon [Mur98] aporta un conjunto de herramientas basadas en la construcción de ITSs[10] [Mur99], donde se proporcionan distintos paradigmas para la gestión del dominio, la estrategia de enseñanza, así como del modelo del usuario y de la interfaz, todas ellas independientes del dominio de aplicación.

2.6 La red semántica

Como ya se ha mencionado, uno de los grandes problemas a los que se enfrenta la web en nuestros días es la falta de estructura, así como establecer y separar claramente la presentación de los contenidos para facilitar el procesamiento automático de la información. Ya se ha visto cómo el mantenimiento de la hipermedia dinámica no es un problema fácil de resolver, sobre todo si tenemos en cuenta la variedad de sistemas y lenguajes de especificación a los que un diseñador debe de enfrentarse de cara a la autoría.

En términos generales, podemos decir que la web está actualmente evolucionando, desde tener inicialmente, y de forma exclusiva, páginas HTML escritas

[10] Intelligent Tutoring System.

o diseñadas a mano, hacia disponer, en un futuro cercano, de un medio donde lo importante en este caso es el procesamiento automático de los datos a partir de una descripción semántica de los contenidos y de los servicios [FFJ+02]. Esto es lo que se conoce comúnmente por el paradigma de la *web semántica* o *red semántica*[11], que consiste en la evolución, apenas iniciada, hacia una web de nueva generación formada por recursos formalmente estructurados y enriquecidos con información semántica explícita, en contraste con la suma caótica de documentos y servicios de la web actual. Para poder estructurar los contenidos de forma adecuada se aplica el concepto de *ontología*, la cual permite añadir información semántica al conocimiento del dominio. Una ontología es una representación conceptual compartida del conocimiento [Gru93a], [Gru93b], lo que en términos prácticos se traduce en una jerarquía de clases interrelacionadas que se utilizan para estructurar el conocimiento en forma de redes semánticas de instancias de las clases junto con axiomas que restringen el modo en que se pueden interpretar las relaciones.

Este paradigma ha sido llevado a término al implementar PEGASUS, el sistema de generación de páginas web dinámicas descrito, que pretende llevar a cabo una separación estricta entre los contenidos (modelo del dominio) y la presentación (modelo de la presentación). Esta característica es aprovechada en gran medida por DESK, permitiendo una autoría de ambos modelos por separado, siendo misión de esta última herramienta el diferenciar qué cambios deberán llevarse a cabo en los contenidos o en los elementos de la presentación, utilizando para ello información de alto nivel disponible en los modelos subyacentes de PEGASUS. De forma similar, la utilización de información del dominio para la caracterización de cambios y de semántica embebida para la localización de estructuras de la interfaz, son acciones que DESK lleva a término gracias a la separación entre los elementos de la presentación con respecto al modelo del conocimiento.

2.6.1 Separación entre contenidos y presentación

Una de las motivaciones que llevó a la evolución de la web, fue sin duda la necesidad de separar los contenidos de la presentación. Este es un aspecto crucial bajo la filosofía de la web semántica, lo que dio lugar a la aparición de lo que se llamaron *documentos virtuales* [MVW99]. Estos documentos contienen elementos que se determinan dinámicamente a partir de pedazos de información almacenados a través de la web, lo que permite almacenar la información de forma ideal para su procesamiento distribuido en función de las necesidades de uso.

Con la evolución de la web surgen nuevos lenguajes como XML, ASP [ASP], JSP [JSP], o XSL [XSL] que aportan, a diferencia del HTML utilizado inicialmente, un lenguaje de especificación más potente para la representación de la información,

[11] Del inglés: Semantic Web

pudiendo así modificar por separado aspectos como el diseño de página y estilos a la hora de componer documentos web. Estos lenguajes surgen como alternativa de otros, como los CGI's y Servlets, que permiten un mantenimiento y manipulación más complejos. A ese respecto XSL es muy adecuado para la generación de HTML a partir de documentos XML, aunque tiene limitaciones importantes en cuanto a que no permite definir estructuras de control avanzadas o, por contra, resultan engorrosas de definir. Otros lenguajes como JSP y ASP permiten embeber código en otros lenguajes de programación, como Java y Visual Basic respectivamente.

2.6.2 Retrospectiva

La web semántica es un paradigma de novedosa creación, cuya reciente historia está ligada estrechamente a otros paradigmas o campos de investigación como la Inteligencia Artificial, en concreto a lenguajes de lógica de predicados, caso éste de algunos de los primeros sistemas basados en ontologías como Cyc [LG90] y KIF [Gen91]. A partir de estos sistemas se desarrollaron otros lenguajes, como Ontolingua [FFR96], basados en KIF, y Frame Logic [KLW95]. Más adelante, el concepto de ontología es llevado al campo de la web y se convierte en pieza clave de la denominada web semántica [BHL01], surgiendo un propósito de estandarización promovido por el consorcio W3C [W3C], y creándose nuevos lenguajes encargados de definir una estructura para la web, como XML [XML], RDF [RDF], OIL [FHH+00], DAML [DAML], [HM00], y SHOE [HH00].

2.6.3 Áreas de aplicación

Uno de los principales campos de aplicación de la web semántica es el de los negocios. Sistemas como WebCADET [CC00] permiten, a través de la interfaz de la web, dar soporte a decisiones mediante el uso de ontologías y aplicando un motor de inferencia. Otros sistemas, como PlanetOnto [DM00], permiten gestionar noticias en grupos de trabajo de forma inteligente. Así mismo, existen otras aplicaciones de propósito más general, como las que ofrecen modelos conceptuales para la manipulación de conocimiento distribuido en grupos académicos (p.e. C-Web [CWe] e IBROW [BF98]). Desde un punto de vista más general, OntoWebber [JDW01] es una aportación que permite el mantenimiento de sitios web a partir de ontologías donde el diseñador puede cambiar elementos de la presentación como estilos y diseños de página predefinidos.

Otra aplicación de los sistemas basados en ontologías es la de la búsqueda automática del conocimiento en la web, como por ejemplo en Ontobroker [Ont], que aporta un lenguaje de especificación para generar esquemas conceptuales de conocimiento en la red, además de un motor de consulta.

Otra aplicación de las ontologías y de la web semántica es la reciente creación de lenguajes y vocabularios compartidos y consensuados que permiten intercambiar conocimiento sobre distintos dominios. A ese respecto existen aportaciones como Dublin Core [Dub] o vocabularios más especializados como Chemdex [Che]. Bajo el dominio del comercio electrónico han surgido aportaciones como B2B [TBP02], WordFNet [Fel99] para el procesamiento de lenguaje natural, KA2 [SAD+00] para el acceso a fuentes de información *on-line* de distintas categorías, o IMS [IMS] y EML [EML] en el campo de la enseñanza

2.6.4 Lenguajes de definición de ontologías

La visión propuesta por Tim Berners-Lee (Figura 2.1) para la construcción de las distintas capas que forman los lenguajes de la red semántica, tienen como base XML como lenguaje de especificación. A partir de esta base se crean nuevos lenguajes, como RDF o RDFS que definen estándares para la representación de metadados, haciendo la información, a partir de este punto, más procesable desde el punto de vista computacional. Los siguientes pasos están todavía por desarrollar o estandarizar, aunque existen algunas iniciativas en marcha al respecto [PF02]. A continuación se describen algunos de los lenguajes más importantes utilizados hoy en día como estándares para la red semántica:

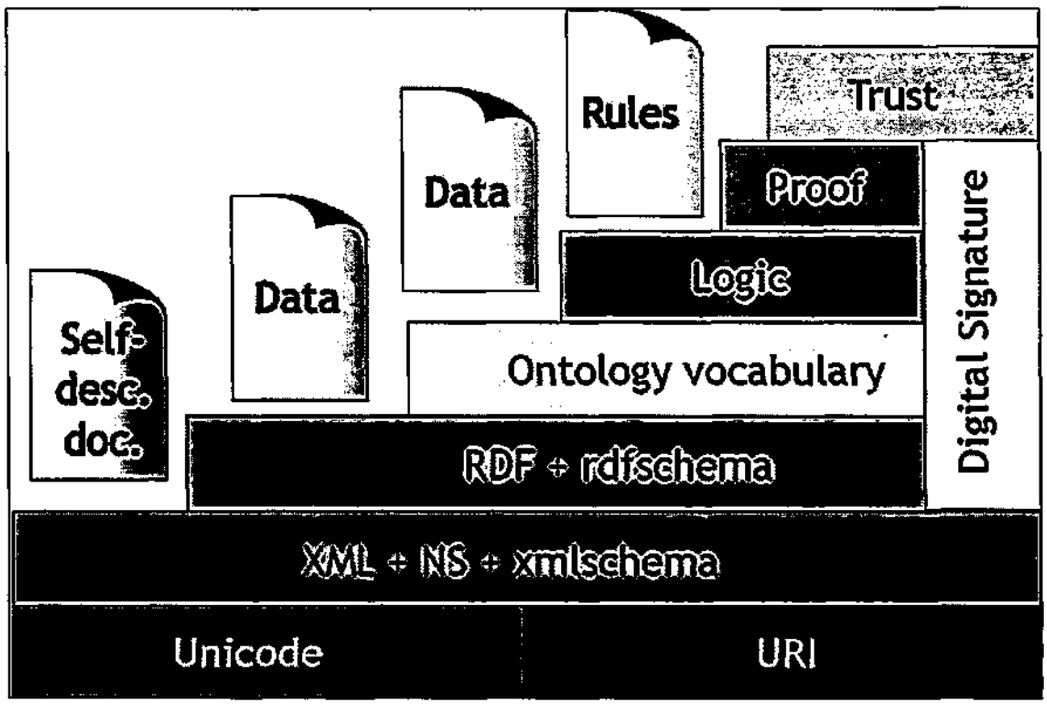


Figura 2.1: Lenguajes para la red semántica

- RDF [RDF] proporciona una sintaxis estandarizada basada en XML para definir ontologías de manera distribuida, incluyendo la noción de clase, instancia y jerarquías de clases. Fue creado por el consorcio W3C, y es actualmente el estándar con mayor aceptación para la construcción de la web semántica.

- OIL y DAML [DO01] están basados en RDF, añaden a este lenguaje la posibilidad de combinar clases y establecer restricciones sobre relaciones mediante expresiones *booleanas*. Recientemente estos dos lenguajes han convergido en uno solo bajo el nombre de DAML+OIL.

- SHOE [Hef01] es una extensión de HTML, fue el primer lenguaje basado en ontologías para la introducción de anotaciones semánticas (metadatos estructurados) en páginas web, con idea de ser reconocidas por agentes inteligentes de localización de metadatos en documentos HTML.

- TopicMaps [XTM] es un lenguaje de especificación que proporciona un modelo gramatical para la representación de la estructura de recursos de información utilizados para definir *tópicos*, así como la relación entre los mismos. Los tópicos se definen como unidades abstractas de información que pueden tener atributos como el nombre, recurso, relación, etc. Las características de los tópicos se definen dentro de un ámbito o campo, es decir, se limita a los contextos dentro de los cuales los nombres y recursos son conocidos por su nombre, recurso y características de relación. A uno o más documentos interrelacionados que emplean esta gramática se le denomina mapa de tópicos o *topic map*.

- OCML [DM00] es un lenguaje de modelado que soporta la construcción de modelos del conocimiento a partir de varios tipos de construcción. Esto permite la especificación y operacionalización de funciones, relaciones, clases, instancias y reglas. OCML también incluye un mecanismo para la definición de ontologías y métodos de resolución.

A parte de los lenguajes mencionados, existen otros trabajos o enfoques que persiguen objetivos más concretos, como es el caso de los lenguajes de modelado para la web bajo el dominio educativo:

- IMS [IMS] es un lenguaje de especificación que facilita la localización *on-line* de actividades educativas distribuidas, como el uso de contenido educativo, el estudio del progreso y rendimiento del aprendizaje, así como el intercambio entre información administrativa sobre los estudiantes.

- EML [Kop00], implementado utilizando XML y SGML, es un lenguaje orientado a la codificación de *unidades de estudio* que se componen de cursos, componentes del curso y programas de estudio de una forma integral. EML no sólo permite describir el contenido de una unidad de estudio (exámenes, tareas, asignaciones), sino

también los roles, relaciones, interacciones y actividades de estudiantes y docentes. EML es neutral con respecto a la estrategia pedagógica utilizada y al soporte de los contenidos, siendo relevante únicamente la forma en que varios elementos educativos se relacionan entre sí a la hora de ser interpretados de forma automática.

- PALO [MVR+99], [RVM+99] es un lenguaje derivado de SGML cuya utilidad apunta a la definición de plantillas instruccionales para organizar material educativo con un propósito específico, aportando una especificación de alto nivel para la navegación e interacción con el usuario mediante la creación de perfiles del estudiante, teniendo en cuenta su rendimiento durante el proceso de aprendizaje. PALO permite instanciar distintos tipos de documentos y realizar referencias a objetos del dominio. Los documentos son compilados y las referencias se resuelven en tiempo de compilación mediante la inserción de los objetos correspondientes para obtener finalmente una página HTML.

Existen otros lenguajes de propósito más general, pero no basados en un estándar definido. Estos lenguajes han sido relevantes por sus aportaciones iniciales al campo de la red semántica, como es el caso de Dublin Core [Dub], un ambicioso proyecto para la creación de vocabularios de meta-datos especializados para la descripción de recursos que aporten mayor información o inteligencia sobre distintos sistemas de información. Otros trabajos, como DPML [SSD98], aportan lenguajes de especificación para la presentación de datos en páginas web. DPML es una extensión de del lenguaje HTML que incluye una serie de construcciones para especificar cómo presentar una o más instancias a partir de una clase determinada, construyendo de esta forma presentaciones personalizadas de los datos dependiendo del perfil del usuario y de la plataforma de acceso. La idea es que un usuario pueda extender la especificación HTML con expresiones como valores de objetos o meta datos, detectados mediante un agente especializado denominado Agente de Presentación (PA). El agente procesa el fichero DPML, sustituyendo las expresiones especiales encontradas por sus correspondientes valores del dominio.

2.6.5 Herramientas

Al igual que se comentó en apartados anteriores, el problema de la autoría es inherente siempre que se trabaje de forma explícita bajo cualquier especificación. En la red semántica esto no es una excepción, y después de la aparición de los distintos lenguajes y estándares se hizo necesario el idear cauces que permitieran un acceso uniforme y flexible, para un usuario final, de cara a la utilización y el mantenimiento del conocimiento compartido [MNV02] en forma de ontología. Algunos de los sistemas y herramientas más importantes, de cara a la manipulación de ontologías del conocimiento para de la red semántica, son los siguientes:

- Ontolingua [FFR96] y Chimaera [MFR+00] proporcionan un entorno distribuido y colaborativo para la creación, edición, modificación, navegación y utilización de ontologías mediante la web [RFP+96]. Ontolingua está basado en un servidor que soporta una alta actividad por parte de los usuarios en la utilización de conocimiento compartido a través de distintas ontologías. El KSL[12] tiene diversos proyectos basados en este servidor de ontologías, como CommerceNet, InterMed, SHADE, Trial Bank, etc. Chimaera además aporta un entorno de diagnóstico para la fusión y chequeo de ontologías de gran tamaño.

- Protégé [NSD+01] es una herramienta de autor que permite al diseñador construir una ontología del dominio y diseñar formularios personalizados para la adquisición de conocimiento. Así mismo la herramienta es una plataforma que puede ser ampliada con widgets gráficos para tablas, diagramas, u otro tipo de componentes para acceder a aplicaciones embebidas. Protégé además incorpora una librería que otras aplicaciones pueden utilizar para acceder y visualizar conocimiento base.

- OILEd [BHG+01] es una herramienta de autor para la edición de ontologías utilizando el lenguaje de especificación DAML+OIL. La idea inicial de OILEd fue la de proporcionar al usuario un editor muy simple para demostrar el uso del lenguaje OIL. Esta herramienta está en desarrollo, y todavía no permite la edición de ontologías de gran tamaño, así como operaciones de migración e interacción entre las mismas.

- WebOnto [Dom98] es una herramienta colaborativa en forma de Applet Java que permite a los usuarios navegar y editar modelos de conocimiento sobre la red. WebOnto proporciona un entorno de manipulación directa de representaciones ontológicas, y es utilizado comúnmente en otros sistemas como PlanetOnto.

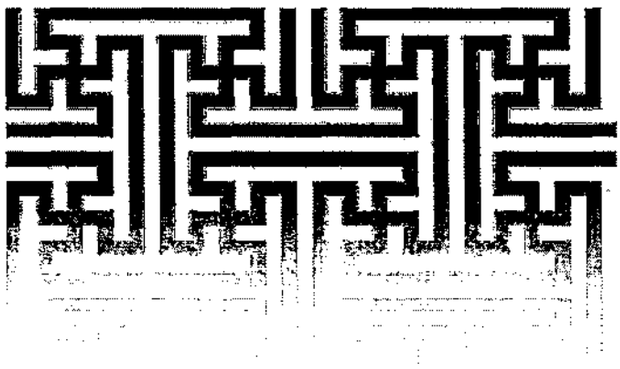
- OntoEdit [SEM00] es una herramienta que permite la inspección, navegación, codificación y modificación de ontologías, soportando de esta forma el mantenimiento y el desarrollo de las mismas. En OntoEdit las ontologías se modelizan con independencia de un lenguaje concreto de presentación, utilizando una interfaz gráfica de usuario para representar distintas vistas sobre estructuras conceptuales (conceptos, jerarquía de conceptos, relaciones, axiomas) en vez de la codificación de dichas estructuras directamente en ASCII. El modelo conceptual de una ontología es almacenado internamente utilizando un potente modelo ontológico, el cual puede ser extrapolado a diferentes lenguajes de representación.

[12] Knowledge Systems Laboratory de la Universidad de Standford

2.6.6 Minería de datos en la web semántica

La minería de la web semántica persigue la combinación de dos áreas de investigación como son, por un lado, la web semántica y, por otro, la minería web ya abordada en el Apartado 2.2.4 de esta tesis. El objetivo que se persigue es el de mejorar los resultados de la minería web mediante la explotación de las nuevas estructuras semánticas de la propia web, así como hacer uso de la minería web, por otro lado, para la construcción de la web semántica. Una fuerte integración de estos dos campos podría incrementar la comprensión de la web para las máquinas, y hacer posible las bases para el desarrollo de un mayor número de herramientas inteligentes para la web [BOS02].

La utilización de la semántica implícita puede ser aprovechada por la minería web para distintos propósitos, siendo la minería de contenidos el área de mayor repercusión, basándose en la codificación explícita de semántica en los contenidos. Este paradigma es de muy reciente creación, por tanto no se sabe con seguridad hacia dónde irán dirigidos los esfuerzos en un futuro no muy lejano, aunque parecen coincidir con la subdivisión y áreas de aplicación explicadas anteriormente, relacionadas con la minería web. De esta forma se puede hablar igualmente de minería de estructura web, así como de minería de uso web. En el caso de minería de uso web se puede explotar la semántica de las páginas web visitadas durante el camino seguido por el usuario, mejorando considerablemente los resultados de la minería web y ayudando al análisis y comprensión de qué es lo que los usuarios están buscando exactamente, qué contenido co-ocurre, etc. La manera más básica de interacción es el uso de ontologías diseñadas ad-hoc en combinación con esquemas automáticos para permitir la clasificación de un gran número de páginas web albergadas en un mismo servidor. De igual forma, la minería de uso puede ser mejorada notablemente si la semántica se embebe explícitamente en las páginas mediante la referencia a conceptos de la ontología, salvando los posibles problemas de redundancia y eficiencia que pudieran existir [HP02].



Capítulo III. Generación de Documentos Web Dinámicos

En este capítulo se describirá el mecanismos empleado para la generación de páginas web dinámicas. Este mecanismo se ha englobado dentro de un sistema de generación de páginas web dinámicas llamado PEGASUS, el cual se explicará en detalle. Igualmente se describirá PERSEUS, una herramienta de autor encargada de la creación y mantenimiento del conocimiento (modelo del dominio) utilizado por PEGASUS. Finalmente se introducirá HADES, un portal adaptativo encargado de la integración de las distintas herramientas comentadas a lo largo de esta tesis.

3.1 Introducción

Una preocupación prioritaria presente explícita o implícitamente en todo el trabajo de investigación llevado a cabo en el campo de la hipermedia adaptativa, es la de encontrar una representación apropiada del conocimiento, así como facilitar un mecanismo de reusabilidad eficiente para el tratamiento de los distintos aspectos involucrados en una interfaz web.

Las metas abordadas en este trabajo con respecto a la edición de páginas web dinámicas se apoyan sobre PEGASUS, un sistema genérico de presentación que proporciona un paradigma de especificación sencillo para definir elementos no triviales de una interfaz adaptativa, independientemente de los contenidos. PEGASUS es una herramienta de generación dinámica de páginas web que hace mínimas suposiciones sobre cómo se representa el conocimiento, con el objetivo de poder ser utilizada con diferentes formas de representación del dominio, y por lo tanto con diferentes sistemas de construcción y gestión de presentaciones web. Por otro lado PERSEUS, como herramienta de autor para la definición de contenidos, se complementa con PEGASUS para ofrecer al diseñador final un paradigma de especificación para la definición de aspectos no triviales de la base de conocimiento de una interfaz web, independientemente del dominio. PERSEUS consiste en un entorno de edición orientado a objetos, el cual proporciona generalidad y usabilidad para la construcción de diferentes tipos de dominios en las presentaciones generadas por PEGASUS.

Por otra parte, el portal HADES está pensado para gestionar los distintos módulos que interactúan entre sí para poder prestar un servicio integrado de autoría al usuario final. Estos módulos abstraen distintas funcionalidades provenientes de otras herramientas, como PEGASUS, PERSEUS y DESK, y consisten en, por ejemplo, la gestión de perfiles de usuario, la generación de plantillas por defecto para la presentación y, en general, en géstionar la información utilizada conjuntamente por cada una de las herramientas.

3.2 PEGASUS

PEGASUS (Presentation modeling Environment for Generic Adaptive hypermedia SUpport Systems) [MC01c], [MC01d], [CM01a], [CM01b], [CM02a], [CM02b] soporta la definición de ontologías del dominio a la medida, para la descripción y estructuración conceptual del conocimiento. Una vez que se ha definido una ontología, el diseñador construye la interfaz web creando objetos del dominio y relacionándolos entre sí utilizando el vocabulario conceptual definido por la ontología. La composición de las páginas a generar se define mediante un modelo explícito de la presentación donde se asocian presentaciones a clases y relaciones de la ontología. La

utilización de un modelo abstracto de la presentación, separado de los contenidos, permite configurar la interfaz adaptativa de contenidos independientemente de la elaboración de los mismos.

3.2.1 Representación del conocimiento

Las diferentes formas de representar el conocimiento dan como resultado un modelo del dominio utilizado como estructura de referencia para mantener un modelo actualizado del conocimiento y objetivos del usuario en relación con el dominio descrito (modelo *overlay*). Esta información se utiliza en tiempo de ejecución para adaptar al usuario la selección y presentación de contenidos y enlaces.

PEGASUS permite la generación automática de documentos hipermedia, con pleno control para el autor sobre los aspectos visuales (presentación) de los documentos generados, y sin imponer una representación particular del conocimiento. Para ello, PEGASUS permite la definición de taxonomías a la medida del dominio y del autor. La terminología así definida se utiliza por un lado para la descripción de la materia por parte del autor, y por otro para la construcción de modelos de presentación asociados a las diferentes categorías del conocimiento (Figura 3.1).

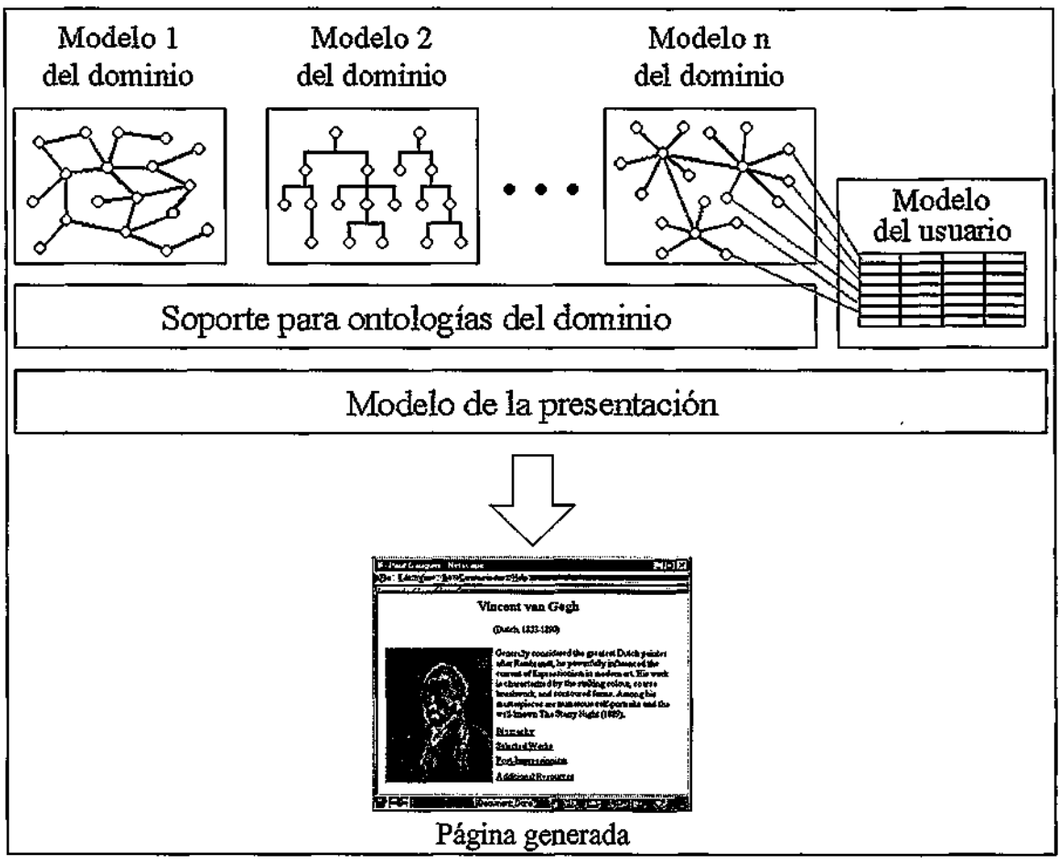


Figura 3.1: Interfaz web basada en ontologías bajo PEGASUS

3.2.1.1 *Ontología*

Como ya vimos en el capítulo anterior, una ontología es una representación conceptual compartida que proporciona una comprensión común de un dominio. Las ontologías se han utilizado comúnmente, bajo el dominio de la web, para la extracción inteligente de conocimiento, como instrumento para modelizar información semántica (metadatos) que se utiliza para anotar documentos web (ver por ejemplo [FAD+99]). En PEGASUS, las ontologías se utilizan para proporcionar la máxima flexibilidad en la representación del conocimiento sobre el dominio. Son además un elemento esencial para conseguir la separación entre presentación y contenidos.

La ontología del dominio en PEGASUS consiste en un conjunto de clases que mejor se adecuan a un campo de aplicación determinado, o que reflejan la visión particular de un autor sobre el dominio. En este enfoque, las ontologías se pueden definir con un alto grado de libertad, con clases muy genéricas, como *Concepto*, *Lección*, o *Hecho*, o más específicas, como *Algoritmo*, *Teorema*, o *Definición*, según el diseñador lo considere más adecuado. Las ontologías incluyen elementos para representar información sobre el dominio (p.e. un teorema tiene un enunciado y una demostración), semántica adicional (p.e. las lecciones tienen niveles de dificultad), e información sobre el estado del usuario y del entorno en tiempo ejecución (p.e. un concepto es conocido o no por el estudiante). Todo este conocimiento se recoge mediante la definición de atributos para las clases, y relaciones entre clases.

PEGASUS proporciona una clase raíz, `DomainObject`, y dos subclases predefinidas, `Topic` y `Fragment`, para ser subclasificadas por los diseñadores. Los tópicos se presentan al usuario final en una página separada, mientras que varios fragmentos pueden ser insertados en la misma página. Una clase predefinida de `Fragment`, `AtomicFragment`, almacena contenido multimedia (código HTML). `DomainObject` tiene algunos atributos predefinidos como `title`, a los que el diseñador puede añadir otros, como `read`, `known` o `visible`, y nuevas relaciones, como `prerequisite` y `subunit`. PEGASUS permite la representación de las clases de la ontología y las instancias del dominio en XML. El siguiente ejemplo es un fragmento que ilustra la definición de una clase Algoritmo con tres relaciones: *procedimiento*, *ejemplos* y *prueba de corrección*.

```
<Class name="Algorithm" parent="Topic">
  <Attribute name="recursive" type="Boolean"/>
  <Relation name="procedure" type="AtomicFragment"
            multivalued="false" title="Procedure"/>
  <Relation name="examples" type="AtomicFragment"
            multivalued="true" title="Examples"/>
  <Relation name="correction" type="Theorem"
            multivalued="true" title="Proof of Correction">
    <Attribute name="relevant" type="Boolean"/>
    <Attribute name="difficulty" type="Number"/>
  </Relation>
</Class>
```

Las relaciones tienen atributos implícitos, con la clase de referencia (`type`) de los objetos que la componen, y el que codifica si la relación se compone de más de un objeto o no (`multivalued`). También pueden tener atributos propios como, en el ejemplo anterior, la *dificultad* de la *prueba de corrección*, que expresan propiedades de la relación en sí. Todas las relaciones tienen un atributo predefinido `title` que se utiliza en algunos casos para generar títulos o texto para enlaces hipermedia.

Además de la ontología del dominio, el diseñador puede definir estructuras de datos más sencillas para describir perfiles de usuario, datos sobre el curso (plan, objetivos, requisitos, duración, nº de alumnos, etc.), características de la plataforma, y otros aspectos considerados relevantes para la adaptatividad del sistema en desarrollo.

3.2.1.2 Modelo del dominio

Una vez definida una ontología, las presentaciones web se construyen mediante la creación de redes semánticas de objetos del dominio, utilizando las clases y relaciones definidas en la ontología. El siguiente ejemplo ilustra la creación de una instancia de *Algoritmo* para representar el algoritmo de Dijkstra para el problema de los caminos de coste mínimo (suponemos que el atributo `title` y la relación `prerequisite` están predefinidos en la clase raíz `DomainObject`).

```
<Algorithm name="Dijkstra" title="Dijkstra's Algorithm"
           recursive="false">
  <prerequisites>
    <Algorithm ref="relaxation"/>                                    (1)
  </prerequisites>
  <procedure>
    <AtomicFragment> <tt>Dijkstra(G,s)<br> Init(G,s)            (2)
    <br> Q = V[G]<br> while Q not empty do<br>
      u = ExtractMin(Q)<br>  for v in
    Adj[u] Relax(G,u,v)</tt> </AtomicFragment>
  </procedure>
  <examples> <AtomicFragment URL="exmpl.html"/> </examples>          (3)
    <correction relevant="true" difficulty="0.6">
      <theorem ref="theorem1"/>
  </correction>
</Algorithm>
```

Los elementos con el atributo `ref` indican referencias a otras unidades del curso (por ejemplo, la línea 1 se refiere a un algoritmo con `name="relaxation"`). Los fragmentos atómicos pueden consistir directamente en un string, como el procedimiento del algoritmo de Dijkstra (línea 2), o una dirección web, como en la línea 3.

En tiempo de ejecución PEGASUS mantiene para cada usuario una copia de todos los objetos del dominio, donde los atributos de las clases (p.e. `read`) se utilizan para medir el progreso del usuario. Estos valores se pueden utilizar para condicionar la presentación, pero su actualización requiere un módulo de actualización complementario (ver arquitectura, Apartado 3.2.3).

3.2.2 Modelo de la presentación

Los sistemas hipermedia adaptativos existentes carecen de un modelo explícito de la presentación. Como consecuencia, la presentación está en parte entremezclada con los contenidos [DC98], y en parte es construida automáticamente por el sistema de acuerdo con decisiones de diseño rígidas que el diseñador no puede configurar [BES98], [CPR99]. En PEGASUS, la separación entre contenidos y presentación se consigue mediante la definición de una *plantilla de presentación* para cada clase de la ontología. Las plantillas determinan qué partes (atributos y relaciones) de un objeto del dominio deben ser incluidas en su presentación y en qué orden, su apariencia visual, y la disposición espacial. Las plantillas se complementan con *reglas de presentación*, que se ocupan de generar elementos de presentación adaptativa que involucran relaciones entre objetos del dominio, a partir de descripciones de alto nivel muy concisas dadas en las plantillas. Así como en otros sistemas, como Eon [Mur98], las interfaces de usuario se asocian con contenidos concretos, en PEGASUS se definen presentaciones para *categorías* de conocimiento.

3.2.2.1 Plantillas

Las plantillas en PEGASUS se definen mediante una extensión de HTML basada en JavaServer Pages™ (JSP) [JSP], que permite intercalar sentencias de control (entre <% y %>) y expresiones Java (entre <%= y %>) en el código HTML. En estas plantillas el diseñador puede hacer uso de todos los elementos de presentación del lenguaje HTML (listas, tablas, frames, enlaces, formularios, etc.), insertando en el mismo, mediante expresiones Java muy sencillas, los elementos del dominio a presentar. En cualquier caso, como se verá más adelante, la idea no es que el diseñador tenga que enfrentarse al lenguaje de modelado de la presentación, sino disponer de una herramienta de autor para que cualquier usuario autorizado no experto en programación pueda editar aspectos referentes a la presentación en PEGASUS. Un ejemplo de plantilla sencilla para la clase *Algoritmo* podría ser la siguiente:

```
<h2> <%= title %> </h2>
<h3> Previous concepts </h3>
<%= prerequisites %>
<h3> Procedure </h3>
<%= procedure %>
<h3> Examples </h3>                                    (4)
<%= examples %>                                        (5)
<h3> Proof of Correction </h3>                         (6)
<%= correction %>                                      (7)
```

En estas plantillas el autor de la presentación sólo tiene que referenciar atributos y relaciones de la clase presentada (en el ejemplo, en negrita). El sistema de presentación se ocupa internamente de aspectos como el manejo automático de listas (relaciones multivaluadas, p.e. los *ejemplos* del *algoritmo* de la línea 5), o la aplicación

recursiva de plantillas a los objetos referenciados según su clase (p.e. los *teoremas* de *corrección* del *algoritmo* de la línea 7). La página resultante para el algoritmo de Dijkstra con esta plantilla de presentación se puede ver en la Figura 3.2. Los elementos HTML que rodean a la presentación del algoritmo (estructura de frames con un índice contextual a la izquierda y botones *Previous* / *Next* al pie) corresponden a la plantilla para la clase raíz `DomainObject`.

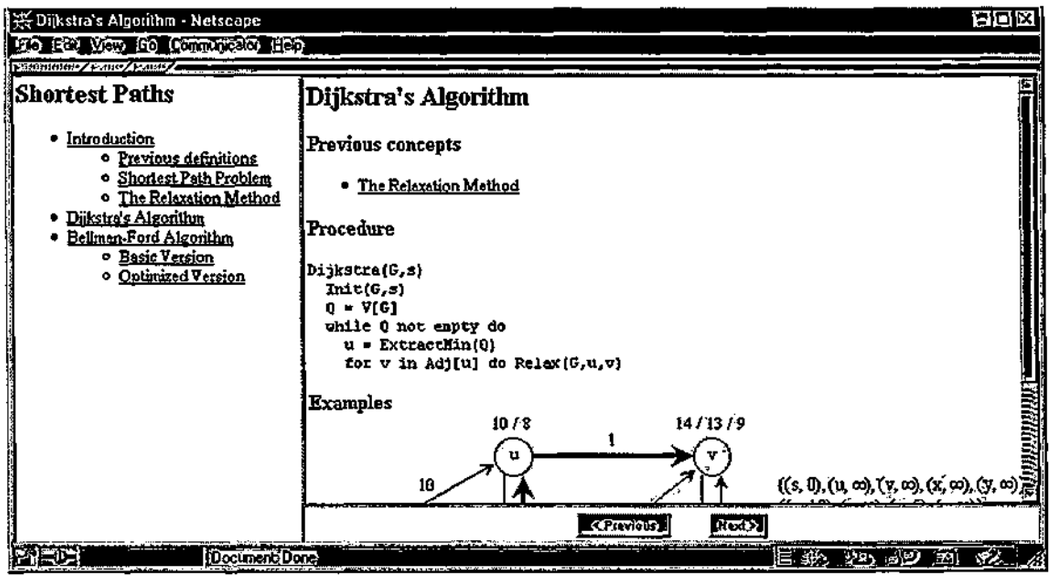


Figura 3.2: Página web generada para un tópico de tipo *Algoritmo*

El lenguaje de definición de plantillas permite introducir elementos adaptativos mediante el uso de condicionales. Así, en las líneas 4 a 7 del ejemplo anterior se podría condicionar la información presentada al nivel de experiencia del alumno, incluyendo todos los ejemplos disponibles cuando el alumno es principiante, y un sólo ejemplo, para alumnos más avanzados, mostrando la prueba de corrección sólo si es relevante y su dificultad no es excesiva para el alumno:

```
<% if (user.expertise < 0.5) { %> <%= examples %>  <% } %>
<% else { %> <%= examples.upto(1) %> <% } %>
<% if (proof.relevant && proof.difficulty < user.expertise) { <h3>
Proof of Correction </h3> <%= correction %> <% } %>
```

El lenguaje de las expresiones incluye así mismo facilidades para, por ejemplo, recortar, filtrar u ordenar listas de acuerdo con una función de comparación arbitraria, generar árboles y listas encadenadas recorriendo una relación, o forzar la generación de enlaces. El uso básico del lenguaje de las plantillas permite especificar un amplio conjunto de presentaciones no triviales mediante una sintaxis muy sencilla. Sin embargo el diseñador puede complicar el lenguaje tanto como quiera escribiendo código Java arbitrario dentro de las propias plantillas.

3.2.2.2 *Reglas*

Las reglas de presentación gobiernan aspectos como la generación de enlaces, la correspondencia entre estilos de enlace y estados de objetos del dominio, la ordenación y disposición espacial de listas (de enlaces o fragmentos), y la generación de presentaciones predefinidas para subconjuntos de la red semántica como árboles y listas encadenadas. Cuando, como en el apartado anterior, el diseñador hace referencia a una relación como `prerequisite` en la plantilla de la clase *Algoritmo*, las reglas se ocupan automáticamente de decidir si se insertan los detalles de cada prerrequisito en la página generada, o si se genera un enlace para cada uno, qué estilo y anotación se utiliza en este caso, y cómo todas las piezas se colocan espacialmente. En este proceso, se analiza si la relación es simple o múltiple, la clase de los tópicos o fragmentos involucrados, su estado, y otras condiciones, en su caso, establecidas por el diseñador.

Las reglas consisten en una lista de cero o más condiciones, seguidas por la presentación a aplicar cuando éstas se cumplen, descrita con la misma sintaxis de las plantillas. Por ejemplo, la siguiente regla establece un determinado color para los enlaces a objetos (unidades de conocimiento) que ya han sido leídos y cuyos prerrequisitos son todos conocidos por el usuario.

```
<Rule class="DomainObject">
    <test condition="asLink && read"/>
    <test-every var="item" list="prerequisites"
                condition="item.known"/>
    <presentation>
       <font color="#006600"> <%= this %> </font>                    (8)
    </presentation>
</Rule>
```

Para que las listas de enlaces se muestren mediante una lista HTML de tipo `<ul>` se puede definir una regla como la siguiente:

```
<Rule class="List[DomainObject]">
 <test-every var="item" list="this" condition="item.asLink"/>
 <presentation>
    <ul> <iterate var="item" list="this">
         <li> <%= item %> </li>                                      (9)
    </iterate> </ul>
 </presentation>
</Rule>
```

Si bien escribir reglas es una tarea delicada que requiere una familiaridad con el sistema, cualquier autor con conocimientos básicos de HTML podría retocar una regla como la anterior para utilizar, por ejemplo, tablas en lugar de listas HTML.

Cuando el sistema de presentación recibe la instrucción de presentar un objeto (como x en la Figura 3.3), antes de aplicar la plantilla correspondiente PEGASUS intenta aplicar todas las reglas existentes. Cuando en la parte derecha de una regla aparecen referencias a objetos (como parte de una relación, o explícitamente como en las expresiones `<%= this %>` y `<%= item %>` de las líneas 8 y 9), éstos son

procesados a su vez, aplicando nuevamente las reglas. Cuando ya no es posible aplicar más reglas, se aplica la plantilla que corresponde al objeto según su clase (*A* en la Figura 3.3); en este sentido, las plantillas se pueden ver como reglas de mínima prioridad cuya condición es *true*. El procedimiento se repite recursivamente con los objetos que a su vez aparezcan al procesar la plantilla (*y*, *z* en la Figura 3.3).

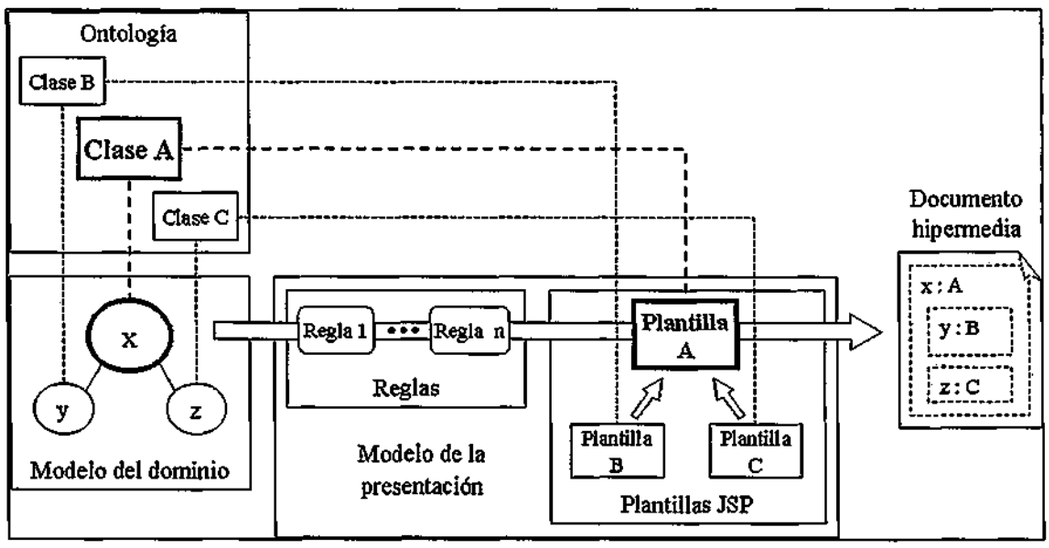


Figura 3.3: Generación de documentos web para unidades de conocimiento utilizando plantillas y reglas

Esta misma filosofía puede utilizarse para la construcción de elementos complejos de la presentación (widgets) que permiten renderizar relaciones multivaluadas de objetos en forma de componentes de la presentación (listas de selección, tablas, combo boxes, etc.). Para ello PEGASUS permite la definición de estructuras personalizadas de la interfaz de una forma similar a como se hace en las reglas, especificando la forma en la que serán visualizados los objetos de la relación y seleccionado el nombre del widget que se desea utilizar para ello.

3.2.3 Arquitectura

En tiempo de ejecución, el usuario interactúa con la aplicación desde un navegador web. La interacción con una aplicación construida con PEGASUS se traduce en un recorrido por la red de objetos del dominio. Cada vez que el usuario se desplaza a un objeto, PEGASUS responde generando una página HTML (Figura 3.4). Para ello, el sistema 1) resuelve la petición del usuario determinando el objeto al que se debe ir, 2) localiza la instancia en el modelo del dominio, 3) actualiza el modelo del dominio y del usuario, 4) genera la interfaz HTML aplicando las reglas pertinentes y la plantilla correspondiente a su clase. En las páginas generadas, los enlaces no corresponden a otras páginas, sino que designan, de forma explícita o descriptiva, objetos del dominio.

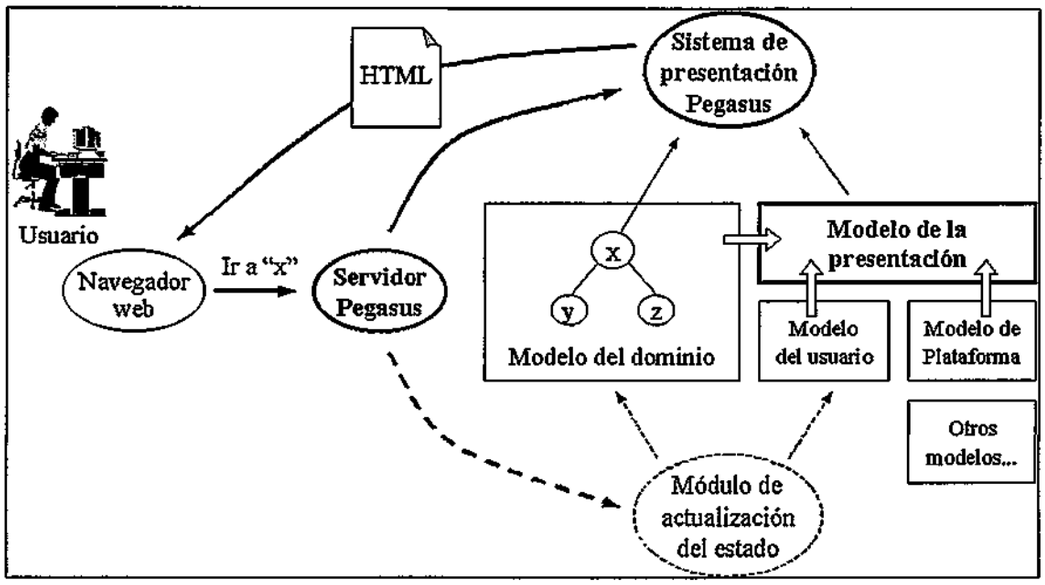


Figura 3.4: Arquitectura de PEGASUS

En PEGASUS, la unidad de interacción con el usuario es pues la petición HTTP. La actualización del modelo del usuario se lleva a cabo teniendo en cuenta únicamente la información transmitida en estas peticiones. Como ya se explicará en el próximo apartado, las características de la plataforma e interfaz de usuario se capturan en el cliente utilizando el portal HADES en la interacción con el usuario final, y esta información es enviada de nuevo a PEGASUS como parte de la petición HTTP cuando el usuario pulsa sobre enlaces y botones de las páginas. Esta suposición simplifica sensiblemente la arquitectura del sistema y la integración con módulos y herramientas externos. A cambio, supone que el sistema no tiene información inmediata y detallada de la actividad del usuario entre dos peticiones, y tampoco se actualiza la presentación en este intervalo. Una granularidad más fina en la interacción se podría soportar generando componentes de interfaz de usuario en Java (applets) que interaccionen con el usuario y se comuniquen directamente con el servidor PEGASUS para consultar y actualizar los modelos del dominio y del usuario.

En general el sistema de presentación no funciona sólo, ya que los pasos 1 y 3 descritos anteriormente son externos a PEGASUS. Una vez que se ha definido una ontología para el dominio, se necesita un módulo que construya y/o actualice las redes de tópicos en tiempo de ejecución (paso 3), como se muestra en la Figura 3.4. Opcionalmente se puede incluir un sistema de planificación como en DCG [Vas97], para determinar el camino a seguir en respuesta a las peticiones del usuario (paso 1).

3.3 Herramientas de autor

Como ya se ha visto, PEGASUS permite la generación de páginas web dinámicas a partir de una ontología y una plantilla de la presentación hechas a medida. Sin embargo, parece razonable presuponer que el usuario final tiene por qué hacer frente directamente al esfuerzo que supone manipular los lenguajes de especificación propios de PEGASUS para la definición de la ontología y del modelo de la presentación.

Esta tesis propone una serie de técnicas, materializadas en distintas herramientas de autor, que permitirán al usuario final interactuar con PEGASUS de una forma más cómoda y sencilla. Estas herramientas de autor, que se describirán a continuación en detalle, comprenden por un lado la edición y el tratamiento de ontologías y de la red semántica de objetos del dominio mediante una herramienta denominada PERSEUS. Por otro lado, y como eje central de la tesis, la herramienta de autor DESK comprende la autoría de presentaciones web a partir de un proceso de ingeniería inversa, de forma que partiendo de una página web generada por PEGASUS la herramienta intenta llegar a los modelos constructores para cambiar el procedimiento de generación de futuras páginas webs del mismo tipo. Otra herramienta denominada HADES permitirá la integración del sistema de generación dinámicas de páginas PEGASUS con el resto de herramientas: PERSEUS y DESK, funcionando como un portal web para controlar el flujo de información entre cada uno de los módulos que componen el sistema de generación y autoría de páginas web dinámicas presentado en esta tesis.

3.3.1 Creación del modelo del dominio mediante PERSEUS

PERSEUS (Presentation ontology buildER for cuStom lEarning sUpport Systems) [MC01e] es una herramienta de autor interactiva para la creación de conocimiento de las presentaciones web generadas por PEGASUS, es decir, para la creación de la ontología del dominio y la red semántica de objetos del dominio (Figura 3.5). La interacción con PERSEUS se desarrolla bajo un entorno orientado a objetos, donde un diseñador puede componer conocimiento del dominio a medida bajo los paradigmas de reusabilidad y facilidad de uso. El modelo del dominio generado por PERSEUS es utilizado posteriormente por PEGASUS como modelo de conocimiento para generar dinámicamente dichos contenidos, según lo relatado en apartados anteriores.

La metodología de trabajo de PERSEUS se basa en tres pasos principales (Figura 3.6): 1) la creación de clases que formarán la ontología del dominio, 2) la instranciación de clases para crear la red semántica de objetos del dominio, y 3) la generación de un fichero XML que reflejará el modelo del dominio completo a utilizar posteriormente por PEGASUS.

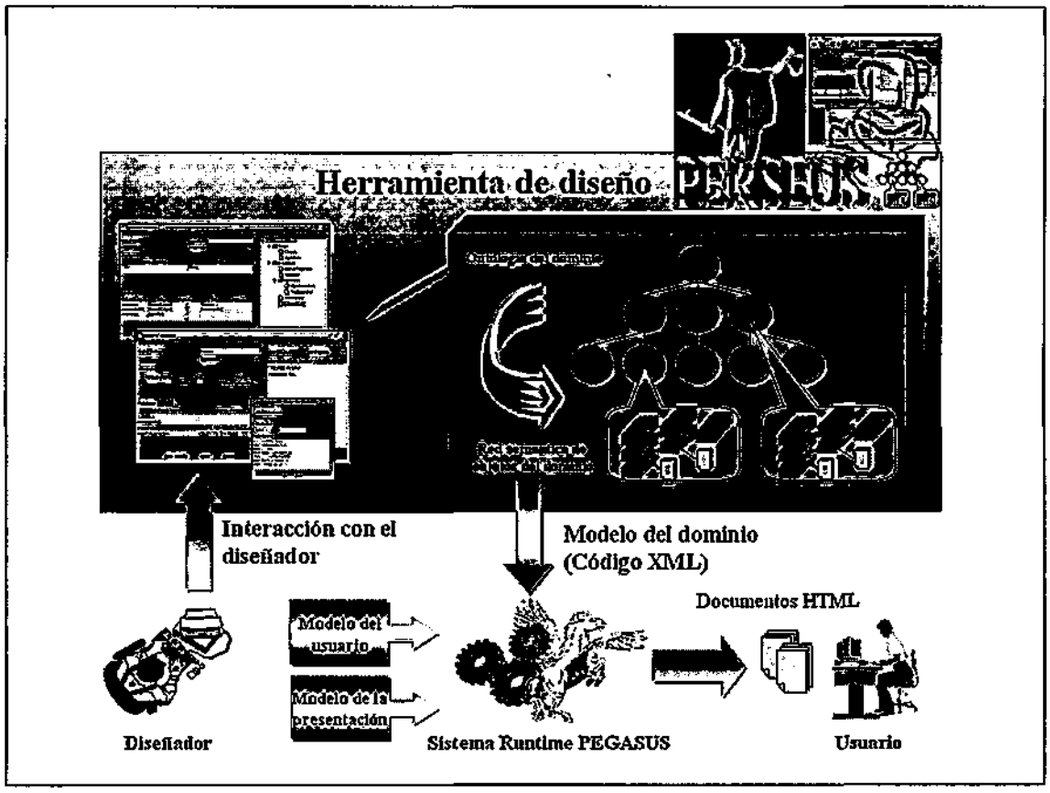


Figura 3.5: Relación entre PERSEUS y PEGASUS para la creación del modelo del dominio de una interfaz web

3.3.1.1 Creación de una ontología del dominio

Utilizando PERSEUS un diseñador puede crear distintas ontologías del dominio, donde cada una de estas se representa en una ventana como la que aparece en la Figura 3.6. PERSEUS, conforme a lo relatado anteriormente sobre la representación del dominio en PEGASUS, posee una serie de clases predefinidas, como `DomainObject`, `Topic`, `Fragment` y `AtomicFragment`, localizadas en la parte superior de la ontología, y que corresponden a las colecciones de clases más abstractas. De esta forma se pueden crear nuevas clases mediante la selección de una clase padre existente, heredando automáticamente los atributos y relaciones ya predefinidas. Para cada clase podemos definir atributos y relaciones, así como atributos de relaciones siendo, el resultado final de este proceso, la creación de una jerarquía de clases (panel derecho de la Figura 3.7) que formarán la ontología del dominio. Mediante PERSEUS el diseñador puede navegar por la jerarquía, modificando y cambiando los valores que estime oportuno y reconociendo el sistema los valores de clases y atributos ya definidos. Como cada ontología se recoge en una ventana distinta, el diseñador puede tener abiertas varias ventanas para así trabajar con distintas ontologías, pudiéndose copiar y mover valores de una a otra dentro del mismo entorno de edición.

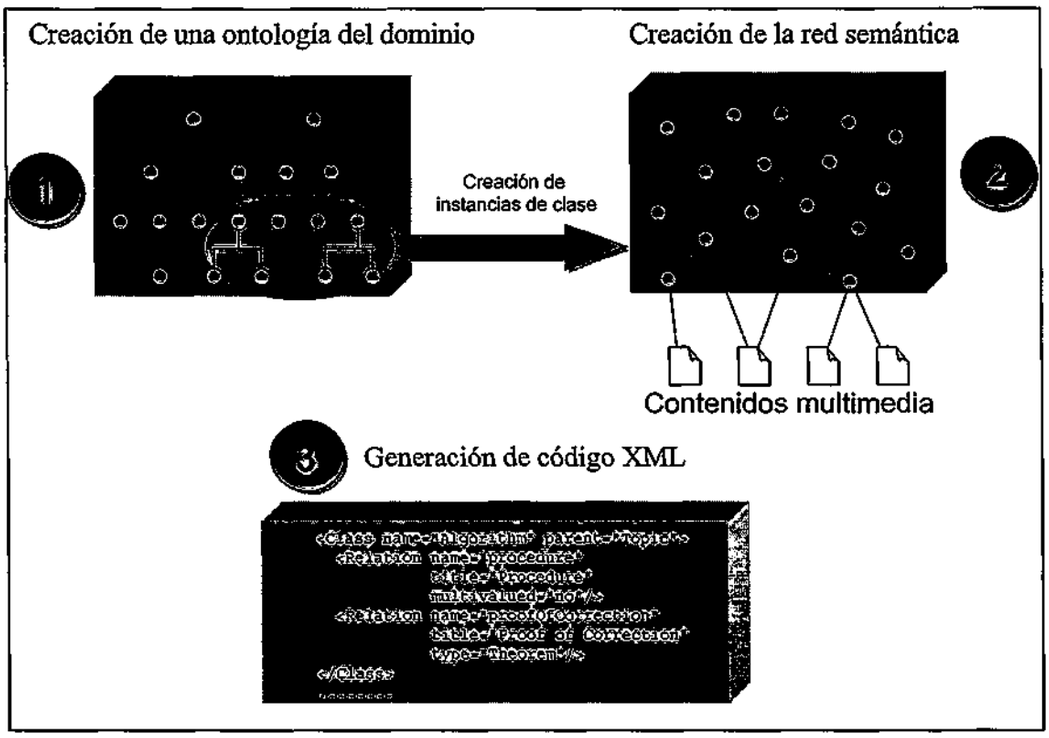


Figura 3.6: Pasos a seguir en la creación de un modelo del dominio bajo PERSEUS

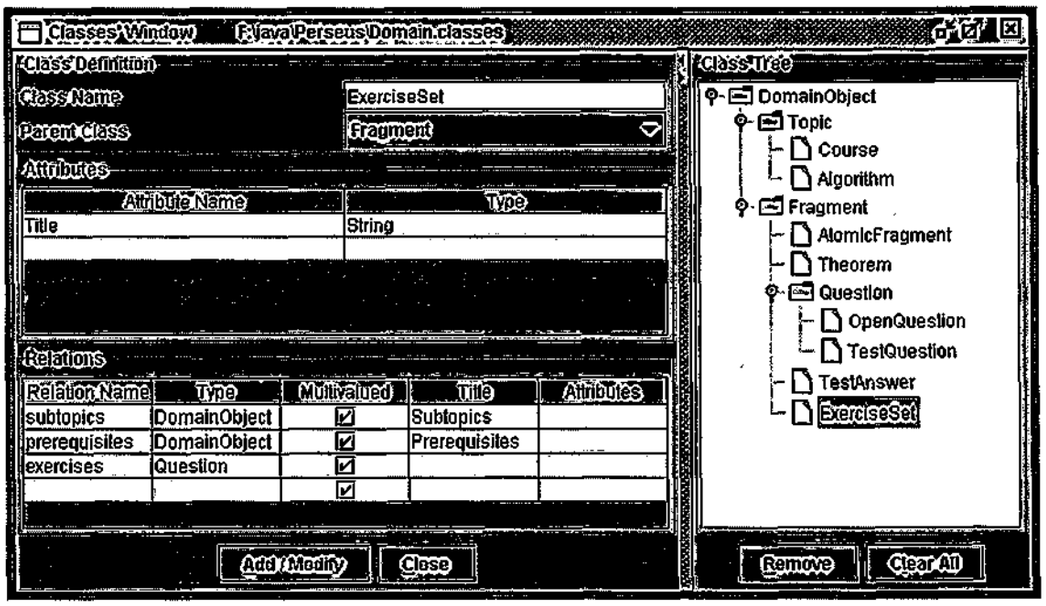


Figura 3.7: Ventana para la creación de una ontología del dominio

3.3.1.2 Creación de una red semántica de objetos del dominio

En PERSEUS, la creación de instrancias de clases de la ontología del dominio (objetos del dominio) se lleva a cabo abriendo una ventana de creación de instancias. De esta forma se puede trabajar a la vez sobre varias instancias pertenecientes a distintas ontologías del dominio al mismo tiempo, moviendo y copiando valores de una a otra. En la Figura 3.8 se muestra una ventana de este tipo, donde puede verse, en la parte derecha, los objetos instanciados (`Dijkstra`, `Graphs`, `Th1`) con sus correspondientes clases (`Algorithm`, `Course`, `Theorem`). Para cada objeto, como es el caso del objeto seleccionado (`Dijkstra`), se pueden introducir valores para sus atributos (`Title=Dijkstra's Algoritm`), así como asociar objetos en las relaciones definidas según la clase a partir de la cual ha sido instanciado el objeto. A partir de las relaciones jerárquicas creadas, se pueden definir fragmentos atómicos (`AtomicFragment`) directamente como componentes de una relación, e introducir directamente un *string* (como es el caso del procedimiento definido directamente en HTML para el algoritmo de Dijkstra en la sub-ventana de la derecha) o incluso asignar un recurso multimedia a partir de su URL. Al igual que sucedía en la ventana de creación de la ontología, el diseñador puede moverse libremente por los objetos creados, incluso filtrarlos a partir de la clase a la que pertenecen, pudiendo modificar unos y otros valores a voluntad.

3.3.1.3 Generación del modelo del dominio

Una vez que han sido definidos, tanto la ontología del dominio como la red semántica de objetos del dominio, el diseñador puede generar el modelo del dominio completo simplemente pulsando un botón en el entorno de edición PERSEUS. De esta forma la herramienta se encarga de generar automáticamente un fichero XML compatible con el formato del modelo del dominio de PEGASUS, para poder construir así el conocimiento de una interfaz web determinada. En la Figura 3.9 se muestra un ejemplo donde se ha generado un modelo del dominio para una interfaz web consistente en un curso sobre teoría de grafos, ejemplo que se ha venido tratando a lo largo del capítulo, tanto en la parte de PEGASUS como en la de PERSEUS.

La información generada sobre el dominio es portable y reutilizable en el sentido de que el diseñador puede hacer persistente el diseño en disco y modificarlo con posterioridad. Igualmente, el formato establecido para la representación del modelo es transportable por la red y visualizable por gran parte de las herramientas y navegadores más utilizados en el mercado. Por otro lado, la representación elegida permite el que un usuario poco experto pueda entender el contenido del modelo fácilmente.

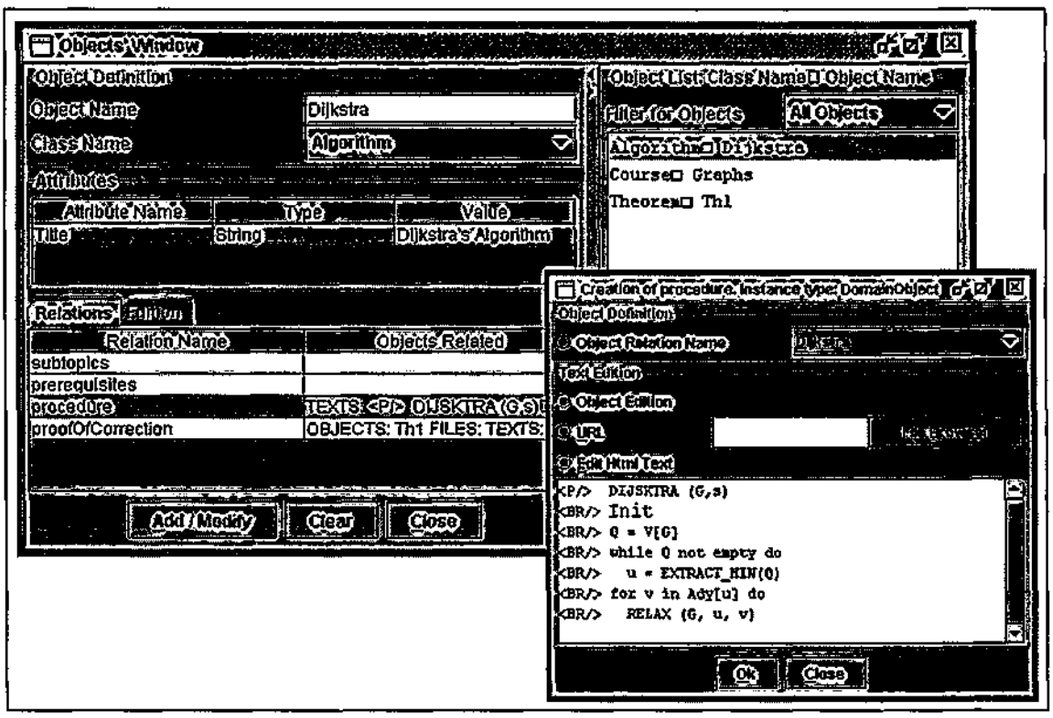


Figura 3.8: Ventana para la creación de una red semántica

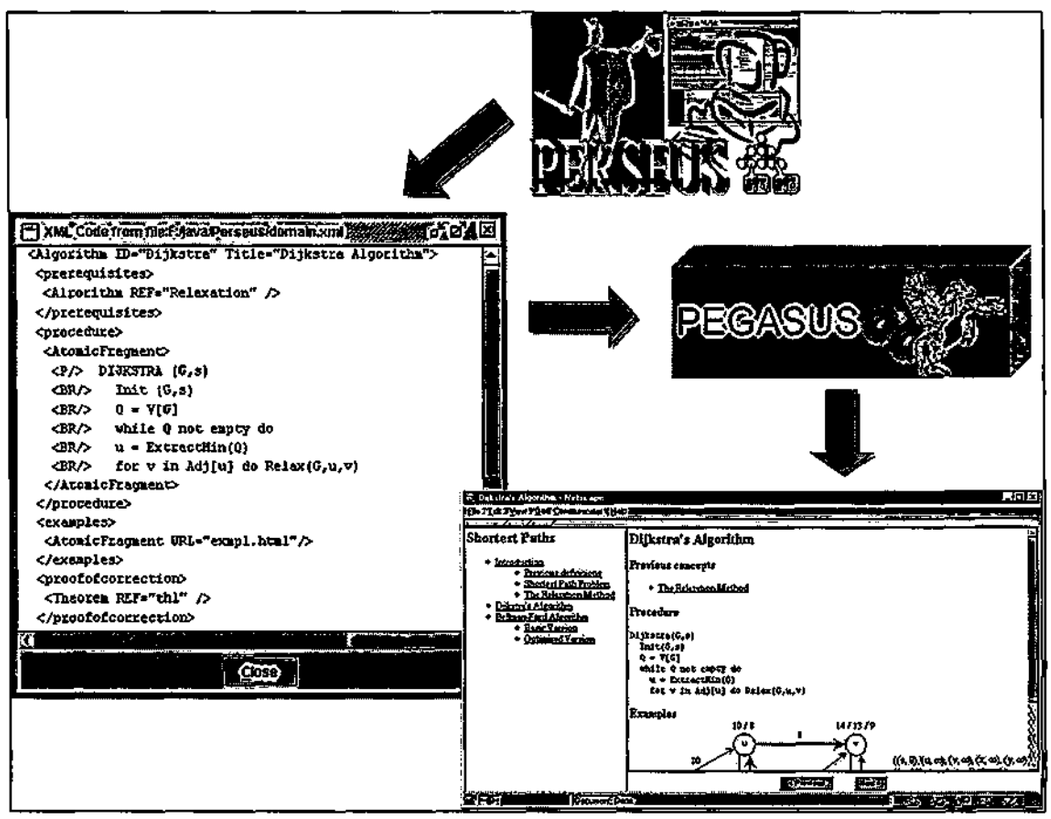


Figura 3.9: Salida de PERSEUS como entrada de PEGASUS

3.3.2 Autoría del diseño de página mediante DESK

A parte de la autoría del modelo del dominio de PEGASUS, se hace latente la importancia de que un usuario final pueda cambiar además la presentación de la página web generada. Esto pone de manifiesto la realidad de enfrentarse a la edición de una página web que ha sido generada dinámicamente, y que por tanto fue creada a partir de información procedural que PEGASUS ejecutó desde el servidor antes de enviar finalmente la página al cliente.

En base a esta necesidad se propone una herramienta de autor: DESK, cuya misión consiste en que el usuario pueda modificar una página web generada por PEGASUS, siendo la herramienta capaz de seguir el camino inverso en la generación de dicha página para poder llegar a identificar los objetos del dominio, presentes en el modelo de la presentación, que generaron la página. De esta forma, DESK cambia información procedural sobre cómo deberá generarse la página modificada en el futuro, o incluso cómo deberán generarse páginas del mismo tipo. Esto es posible gracias a la separación explícita que hace PEGASUS de presentación y contenidos, permitiendo que la edición de esos dos conceptos por separado sea un reto que DESK lleva a cabo de manera automática.

La herramienta de autor DESK es el eje central de esta tesis, por tanto será merecedora de un capítulo completo (Capítulo IV), en el cual se abordará detalladamente la forma de proceder de dicha herramienta, describiendo las heurísticas que permiten llevar a cabo la inferencia de los cambios llevados a cabo por el usuario, así como los mecanismos de desambiguación ideados. De la misma forma se explicará cómo la herramienta implementa un agente de inferencia, ideado para automatizar tareas mediante la detección de patrones de iteración del usuario en su interacción con la herramienta.

3.3.3 Integración mediante HADES

Todas las herramientas vistas hasta ahora permiten, de forma aislada, el hacer frente a la generación y autoría de páginas web dinámicas de una forma asequible para el usuario. Sin embargo, es necesario integrar todas estas herramientas para poder canalizar el esfuerzo de proporcionar al usuario final una única vía de acceso a todos los recursos presentados hasta ahora. La idea es que el usuario pueda abstraerse de la localización de cada herramienta y que éstas puedan darle servicio cuando lo necesite, intentando automatizar ciertas tareas de integración entre la salida de una herramienta y la entrada de otra.

Bajo este planteamiento surge HADES (Hypermedia ADaptive Educational Server) [MC01e]. HADES es un portal adaptativo encargado de la gestión de los distintos módulos de las herramientas descritas, así como de adecuar el flujo de

información de una herramienta a otra. En un principio este portal estaba pensado para ser utilizado con dominios educativos, pero actualmente el portal está preparado para soportar cualquier tipo de interfaz web independientemente del dominio.

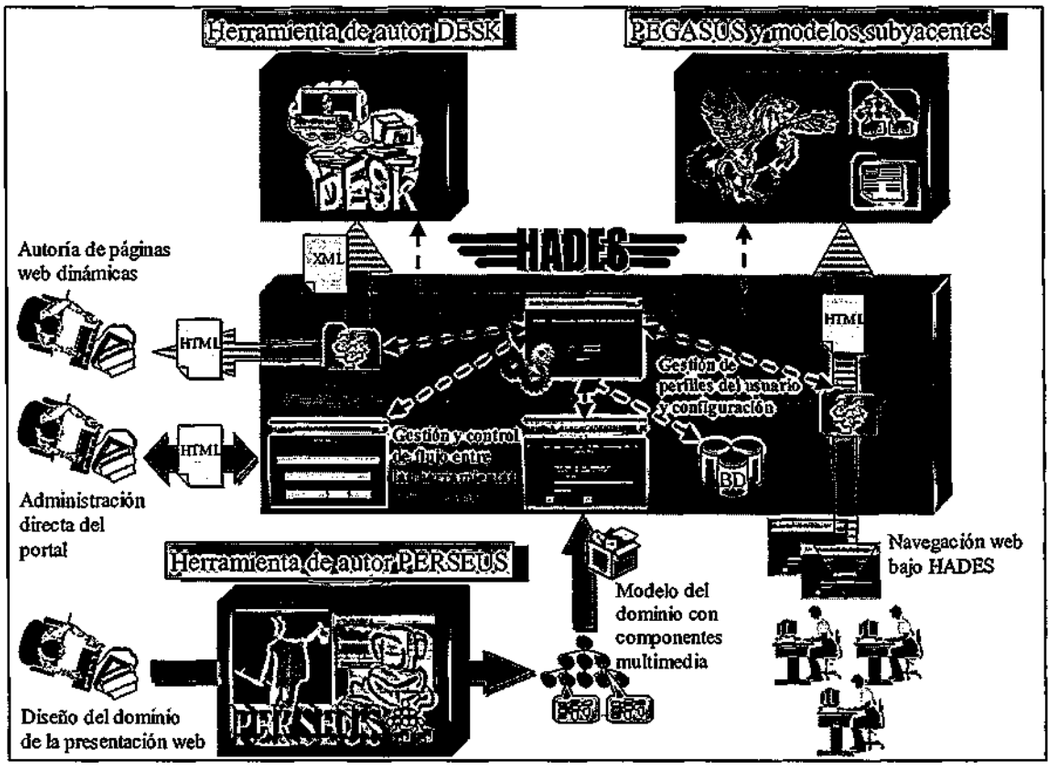


Figura 3.10: Integración de las distintas herramientas con HADES

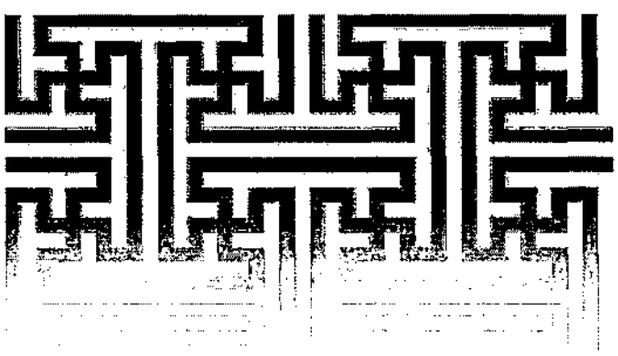
En la Figura 3.10 se muestra la integración de las distintas herramientas a través de HADES, así como los distintos flujos de información que fluyen de una a otra herramienta, pasando siempre por el mecanismo *runtime* central de HADES encargado de gestionar los distintos módulos y hacer la información de la sesión persistente mediante la utilización de una bases de datos. A partir de la figura descrita anteriormente, las tareas básicas que realiza HADES son las siguientes:

- Gestión de los perfiles del usuario. Se realiza una extracción automática de información sobre el usuario, como su localización, el idioma utilizado en la configuración de su navegador, el sistema operativo utilizado así como la versión del mismo. Mediante un formulario el usuario introduce otros datos, como su *login* y *password* en el sistema, además de otros datos personales. La información del perfil del usuario será muy útil a la hora de interactuar con PEGASUS o DESK, que hacen uso de este conocimiento para adaptar la información presentada o para poder llevar a cabo ciertos tipos de cambios en las páginas web dinámicas. Así mismo, HADES permite crear distintos perfiles como *administrador*, *editor* y *estudiante*, para el

acceso a ciertas funcionalidades a partir de cada rol asignado. El perfil de administrador permite cambiar aspectos internos y administrar el resto de perfiles de usuario, cambiando sus propiedades, monitorizando la actividad de los usuarios en el sistema, así como asignándoles nuevos niveles de restricción en el acceso al propio portal y, por lo tanto, al resto de aplicaciones con las que se comunica HADES. Además, el perfil de administrador permite cambiar otros aspectos como, por ejemplo, el formato en el que se generan las plantilla de presentación inicialmente. Por otro lado, el perfil de editor permite llevar a cabo la autoría de presentaciones PEGASUS mediante herramientas como DESK, así como generar la base de conocimiento, mediante PERSEUS, relativa a las páginas web a generar por PEGASUS. Por el contrario, el nivel más bajo de acceso es el de estudiante, que permite únicamente navegar por las distintas páginas web generadas por PEGASUS, sin posibilidad de realizar ningún tipo de modificación.

- Interacción con PEGASUS. Mediante HADES, un usuario autorizado puede enviar la información del dominio (ontología y red semántica) creada con la herramienta PERSEUS, junto con otros ficheros necesarios (componentes multimedia), siendo el propio portal el encargado de crear la infraestructura para dar de alta una nueva interfaz web. Por defecto la primera vez, HADES crea unas plantillas JSP estándar para el modelo del dominio enviado por el usuario, de forma que se crea una plantilla por cada clase de la ontología. En esta plantilla aparecerán los objetos involucrados con un layout y estilo inicial muy sencillos. Posteriormente, el usuario podrá cambiar la presentación utilizando la herramienta de autor DESK. La ventaja es que, realmente, el usuario solo tienen que preocuparse de enviar el conocimiento, encargándose el portal del resto del trabajo. HADES interactúa con PEGASUS para presentar la información al usuario sobre una determinada interfaz web creada inicialmente, de esta forma HADES visualiza la información que PEGASUS genera y además le envía y tiene en cuenta el perfil del usuario durante la sesión.

- Interacción con DESK. HADES permite integrar la interacción del usuario bajo la herramienta de edición DESK junto con la generación de páginas web dinámicas de PEGASUS, permitiendo así un marco homogéneo para la edición y el intercambio de parámetros entre estas herramientas. El perfil del usuario se tiene también en cuenta a la hora de editar la presentación, distinguiendo en este caso si un usuario tiene permiso o no para modificar aspectos de una interfaz web, como los contenidos o la plantilla de la presentación.



Capítulo IV.
Autoría Mediante Ejemplos

En este capítulo se describirá el mecanismo empleado para la autoría de páginas web dinámicas. Este mecanismo se hace efectivo mediante la creación de una herramienta de autor denominada DESK, la cual se explicará en detalle a lo largo de este capítulo.

4.1 Introducción

Se ha estimado que hoy en día el 80% de los documentos web son generados de forma dinámica [SA00]. Esto conlleva una mayor complejidad en la edición de estas páginas que, al ser generadas dinámicamente, no permiten cambiar mediante un editor HTML la información relativa a su generación (i.e. información procedural o procedimientos de generación de dichas páginas). Es por esto que, en general, la autoría de páginas web dinámicas es una tarea nada sencilla de llevar a cabo por un usuario no experto en la programación para la web, lo que hace de dicho problema un aspecto lo suficientemente relevante como para investigar posibles mecanismos que permitan una edición fácil de este tipo de documentos por parte de los autores web. Hasta la fecha las aportaciones a este problema son limitadas. Las herramientas existentes permiten la edición y creación de páginas web estáticas, algo que hace más asequible la creación de documentos por parte de los autores web, pero que no proporciona ayudas para la integración con los sistemas que generan documentos web dinámicos.

Como se vio en el capítulo anterior, una separación explícita entre la presentación y los contenidos ayuda al tratamiento automático de estos dos conceptos por separado, como primer paso para la autoría de documentos que se generan de forma dinámica a partir de este tipo de modelos. A ese respecto, PEGASUS permite una generación de documentos web dinámicos a partir de una base de conocimiento, donde se recoge el conocimiento, y un modelo de la presentación donde se especifica, mediante plantillas y reglas, cómo se renderizará la información del modelo del dominio, construyendo finalmente la página web final que será accedida por el usuario final bajo el navegador. Sin embargo, como ya se señaló, y a pesar de esta separación explícita, no es trivial para un usuario el escribir código en estos lenguajes de especificación, siendo ideal el disponer de algún mecanismo que permita al usuario final abstraerse del lenguaje de modelado propio de la herramienta de generación dinámica como es, en el caso de PEGASUS, el lenguaje utilizado para la construcción del modelo de la presentación.

En base a estas argumentaciones, esta tesis propone un mecanismo para afrontar el problema de la autoría de páginas web dinámicas, materializándose en la construcción de una herramienta de autor inteligente: DESK. El objetivo principal que persigue DESK es el de proporcionar a los autores web una herramienta con una interfaz WYSIWYG para la edición de páginas web dinámicas. Mediante DESK un usuario edita una página HTML concreta generada por PEGASUS. La herramienta analiza los cambios que el usuario está haciendo sobre el documento y, como resultado final, DESK cambia el procedimiento de generación de la página a base de modificar, por parte del usuario, un producto final del propio procedimiento (i.e. la página HTML generada dinámicamente), generalizando además este mecanismo a todos los casos del mismo tipo (i.e. objetos de la misma clase, mirar Capítulo III sobre PEGASUS). Por ejemplo, si tenemos una página web sobre pintura, los cambios en una página sobre van

Gogh, daría como resultado el que todos los pintores que fueran posteriormente mostrados con esa plantilla se vean afectados por dicho cambio.

DESK es una herramienta construida utilizando como base paradigmas como la Programación por Demostración, así como heurísticas propias para afrontar problemas como la desambiguación de acciones del usuario y el análisis de estructuras de diseño de página. Un aspecto diferenciador, respecto a otros sistemas basados en Programación por Demostración, es que DESK utiliza conocimiento procedente de la base de conocimiento (i.e. el modelo del dominio de PEGASUS), caracterizando información relevante [MC03b], y modificando así el procedimiento de generación de las páginas web de forma persistente. Siguiendo bajo el paradigma de la Programación por Demostración, DESK además incorpora un asistente que permite que algunos de estos cambios sean llevados a cabo automáticamente por la herramienta (p.e. convertir una lista de selección en una tabla o viceversa), en concreto los cambios que involucran la reorganización y transformación de widgets (i.e. listas, tablas y controles HTML). DESK detecta *patrones de iteración* sobre las acciones de edición llevadas a cabo por el usuario, e infiere cambios de alto nivel entre distintos controles HTML, conservando la relación semántica de la estructura con los datos del dominio y permitiendo llevar a cabo la sustitución entre el widget origen y destino, reorganizando la copia de elementos de uno a otro widget en función de la secuencia inicial introducida por el usuario. Esto se lleva a cabo mediante la observación de las acciones que lleva a cabo el usuario bajo la herramienta (i.e. monitorización del usuario), técnica por otro lado empleada en algunos entornos de Programación por Demostración (ver Capítulo II).

La forma de trabajar de DESK lleva implícito un proceso de ingeniería inversa, en cuanto a que se parte de algo ya generado (i.e. una página de PEGASUS) y se trata de localizar la información en los modelos constructores iniciales, cambiando la forma en la que se generará en el futuro la presentación. En definitiva, de lo que se trata es de reconocer, en las piezas que forman la página, las entidades del dominio a las que corresponden además de determinar las relaciones entre los fragmentos.

El mecanismo llevado a cabo por la herramienta permite a un usuario final realizar los siguientes tipos de cambios:

- Modificar, eliminar y añadir contenidos dinámicos procedentes de la base de conocimiento.
- Modificar valores atómicos de la base de conocimiento. En este caso y en el anterior, se trata de realizar cambios concretos, pues la creación y edición de la base de conocimiento se utiliza la herramienta PERSEUS.
- Modificar, añadir y eliminar estructuras de interfaz HTML, como tablas y otros controladores.
- Mover, eliminar y añadir fragmentos HTML estáticos.
- Cambiar etiquetas HTML y estilos.

La forma de aplicar los cambios anteriores, una vez detectados y caracterizados éstos, es mediante la modificación del modelo del dominio o de la plantilla del objeto de la página afectada, de modo que estos cambios afectarán a la presentación de cualquier otro objeto de la misma clase.

Al final de este capítulo se verá un ejemplo completo de utilización de DESK en la autoría de una interfaz web generada a partir de PEGASUS. En este ejemplo se tratará, de una forma práctica, cada uno de los conceptos que a continuación se describen, y que tienen que ver con la modelización interna llevada a cabo en el diseño de la herramienta de autor DESK. Estos conceptos incluyen los mecanismos de inferencia utilizados, describiendo las heurísticas involucradas para poder llevar a término la correcta inferencia de cambios llevados a cabo por el usuario sobre una página HTML.

4.2 La herramienta de autor DESK

DESK (Dynamic web documents by Example using Semantic Knowledge) [MC02a], [MC02b], [MC03a], [MC03c], [MC03d] es una herramienta de autor pensada para facilitar al usuario la edición de páginas web dinámicas mediante un sencillo interfaz de usuario muy similar al de un editor web como el que podemos encontrar en aplicaciones comerciales o de libre distribución, como Microsoft Internet Explorer [Mic], Netscape-Composer® [Net] o Mozilla [Moz]. La principal diferencia entre DESK y las citadas herramientas es que estas últimas sólo permiten la edición de páginas web estáticas. Por el contrario, DESK permite la edición de documentos web dinámicos a partir de heurísticas propias que le permiten identificar lo que el usuario quiere llevar a cabo. Para ello DESK localiza partes integrantes de los modelos de la presentación y del dominio de PEGASUS, asociándolas a acciones del usuario para asistir a éste en la creación de estructuras de la interfaz complejas. Otras herramientas comerciales, como las últimas versiones de XMLSpy® [Spy], Macromedia HomeSite [Mac] y Kawa [Kaw], permiten la edición de páginas JSP, encargadas de la generación dinámica de código HTML. Sin embargo, aunque estas otras herramientas incluyen funcionalidades tales como cambiar el color de las partes integrantes del código y la disposición tabular de los elementos, su funcionalidad se limita esencialmente a la edición de código JSP, sin aportar un entorno WYSIWYG y sin distinguir posibles correspondencias entre la base de conocimiento y la página final generada.

Para llevar a cabo su cometido, DESK distribuye su funcionalidad en dos partes construidas a partir de una arquitectura cliente-servidor, tal y como se muestra en la Figura 4.1. La parte del cliente (front-end) es la herramienta de autor WYSIWYG encargada de soportar la mayor parte de la interacción directa con el usuario. La parte del servidor (back-end), es donde se efectúan los mecanismos de inferencia que permiten llevar a cabo los cambios realizados por el usuario sobre la interfaz web. Atendiendo a la información mostrada en la Figura 4.1, los principales pasos en el proceso de interacción con DESK son los siguientes:

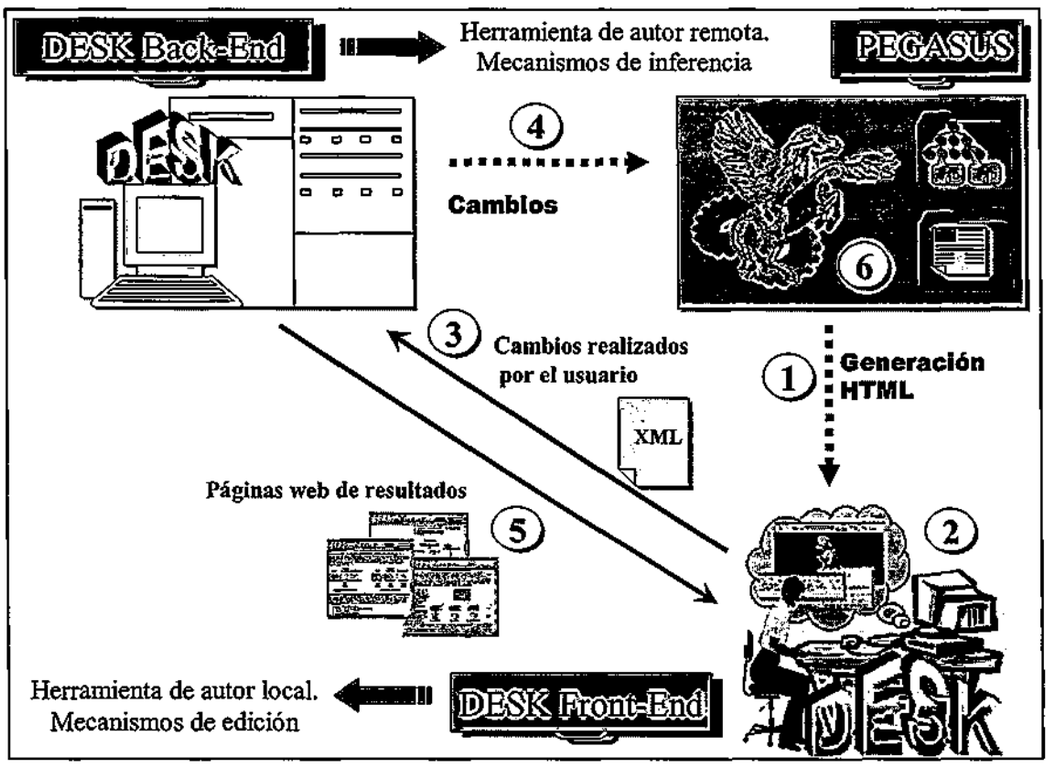


Figura 4.1: Herramienta de autor DESK

1) En primer lugar el usuario edita una página concreta, generada por PEGASUS, mediante la herramienta DESK del front-end.

2) DESK recoge información sobre la interacción del usuario y la plasma en un *modelo de monitorización*, que es un modelo estructurado codificado en XML, el cual refleja información de alto nivel sobre las acciones o cambios llevados a cabo por el usuario durante la edición de un documento web.

3) Los cambios realizados por el usuario se envían al servidor. La parte de DESK que se encuentra en el back-end procesa los cambios a partir del modelo de monitorización, empleando para ello los correspondientes mecanismos de inferencia mediante el uso de conocimiento de alto nivel del dominio.

4) Como resultado final del proceso anterior de inferencia se modifican los modelos subyacentes de PEGASUS: presentación y dominio.

5) Finalmente se envía al usuario información relativa a los cambios efectuados y a lo acontecido durante todo el proceso de inferencia.

6) La próxima vez que se genere una página del mismo tipo, los cambios surtirán efecto, habiendo modificando así el mecanismo por el cual se generan las páginas web de forma persistente.

4.3 DESK cliente

DESK cliente funciona como un editor web genérico, interpretando la información que le llega desde PEGASUS y presentándosela al usuario final mediante una interfaz WYSIWYG adecuada. La misión principal del cliente DESK es la de realizar un seguimiento del usuario durante la interacción, de esta forma se guarda información sobre todo lo relacionado con lo que el usuario hace durante el proceso de edición de un documento, es decir, insertar, borrar o modificar un fragmento de la página, cambiar la distribución de ciertos elementos, modificar los atributos de un fragmento multimedia, crear nuevos objetos de presentación (cuadros y listas de selección, tablas, numeraciones y viñetas), etc.

La Figura 4.2 muestra el aspecto de la interfaz de usuario de DESK, donde el usuario edita una página generada por PEGASUS, y que proviene del ejemplo que se vino tratando en el Capítulo III (Generación de documentos web dinámicos) sobre teoría de grafos, en concreto sobre el teorema de Corrección del Algoritmo de Dijkstra. La figura está dividida en dos ventanas de forma que, en la ventana superior, puede observarse el entorno de navegación y edición de páginas web, y en la ventana inferior se reflejan una a una las acciones que el usuario lleva a cabo al editar el documento (i.e. el modelo de monitorización). Como puede verse en la ventana superior de la mencionada figura, el aspecto de la interfaz de usuario de DESK es similar al de otras herramientas de edición web. De esta forma el usuario dispone de una barra de herramientas para la edición, así como de las acciones características de manipulación de ficheros, ayuda, barra de navegación, etc. La ventana inferior que refleja el modelo de monitorización en XML, se puede mostrar u ocultar mediante una opción del menú *Log*. El modelo de monitorización refleja una representación estructurada de las acciones del usuario durante la edición del documento.

DESK cliente implementa un mecanismo de disección de las acciones del usuario durante la edición. Este mecanismo se refleja en la Figura 4.3, donde se representan los pasos empleados para llegar a la caracterización sintáctica de los cambios llevados a cabo por el usuario. La forma de llevar esto a término, es mediante la localización de la información que rodea a los elementos HTML que se ven afectados por los cambios que acomete del usuario a la página HTML, siendo el mecanismo utilizado el que se detalla a continuación (Figura 4.3):

a) En primer lugar, los eventos de bajo nivel producidos por el usuario bajo el entorno de edición son registrados y analizados, filtrando los que son relevantes y que darán lugar a acciones de edición o cambios del usuario sobre la página HTML.

b) Seguidamente, mediante la aplicación de *heurísticas de bajo nivel* (H_L), se trata de delimitar bloques significativos. Para ello se intenta caracterizar la posición del cambio llevado a cabo por el usuario, almacenando información relativa al contexto

como la posición dentro del bloque, así como los bloques HTML de alrededor. Esta información se encapsula y se representa con primitivas de monitorización.

c) Estas primitivas permiten construir de una forma ordenada y fácil de procesar, lo que finalmente constituirá el modelo de monitorización del cliente que será enviado al servidor para su posterior procesamiento.

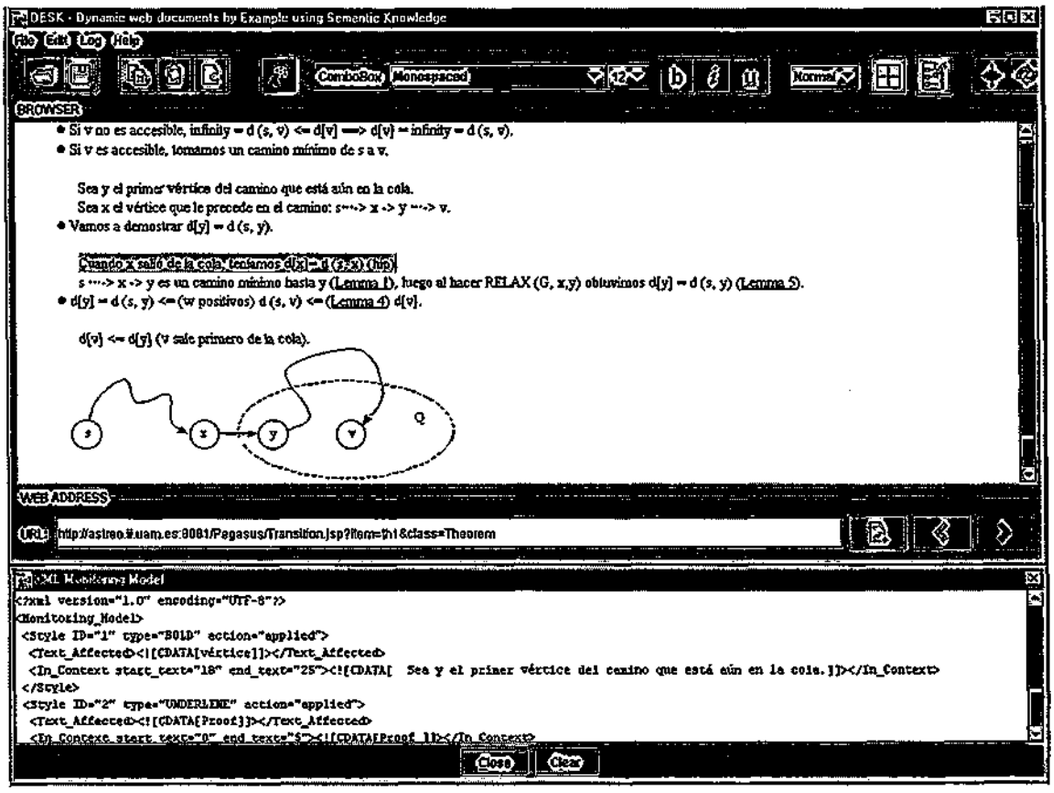


Figura 4.2: Aspecto de la interfaz de usuario de DESK

El objetivo final de esta caracterización sintáctica de cambios es el de tener la máxima información en el servidor a partir de los medios de los que se dispone en el cliente (fragmentos HTML), dado que partimos de una página que ha sido generada dinámicamente. De esta forma será posible asociar estos cambios a objetos del dominio con la menor ambigüedad posible (como se verá en el Apartado 4.4). En los siguientes apartados se detalla en profundidad las partes más importantes del mecanismo de caracterización de cambios en DESK cliente.

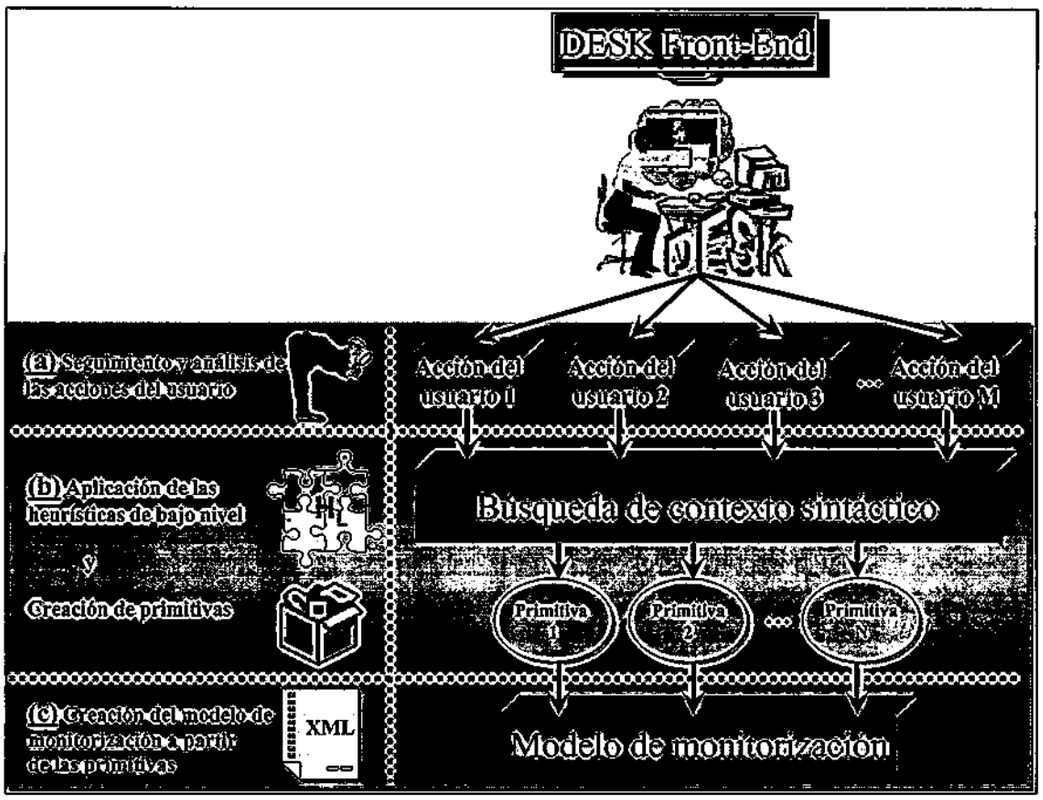


Figura 4.3: DESK cliente

4.3.1 Heurísticas de bajo nivel

Las heurísticas de bajo nivel se componen de una serie de módulos encaminados a localizar el marco de la acción llevada a cabo por el usuario al editar el documento. Estos módulos están formados básicamente por algoritmos de localización de contexto y caracterización del bloque sintáctico en donde se enmarca cada acción del usuario. Básicamente son tres los módulos que componen las heurísticas de bajo nivel, a) un primer módulo encargado de la caracterización y localización del lugar donde se efectúa un cambio llevado a cabo por el usuario, b) otro módulo encargado de la caracterización de estructuras o controles HTML más complejos (como tablas y listas) y, finalmente, c) un módulo de empaquetamiento y construcción de la primitiva en sí junto con los parámetros y atributos utilizados en su codificación dentro del modelo de monitorización. Atendiendo a esta clasificación, la especificación de cada uno de estos módulos es la siguiente:

a) Módulo general de localización de contexto en el cliente.

Este módulo persigue facilitar la localización de los cambios en documentos web a partir de las acciones llevadas a cabo por el usuario. Por cada una de esas acciones se intenta localizar dónde se ha realizado, en qué posición y se estudian también los fragmentos HTML que rodean al texto objeto de la acción. En la Figura 4.4 se muestra un ejemplo donde pueden observarse tres casos de una acción del usuario

correspondiente a la inserción de un fragmento HTML (`Nueva inserción`) y la forma en que este módulo representa la información contextual de dicha acción, (`InContext`) en función del boque HTML (`Párrafo de contexto`) anterior (caso a, `location="after"`), posterior (caso b, `location="before"`) o dentro del propio bloque sintáctico (caso c, `starts=15 ends=40`). La localización se lleva a cabo mediante la caracterización del lugar en el cual se ha llevado a cabo dicha acción del usuario. Para ello se busca el contexto a partir de bloques HTML cercanos, y se codifica esta información utilizando atributos que describen su posición con respecto a estos bloques, en términos de `after` y `before` siempre que sea posible (cuando se encuentren párrafos alrededor) y necesario (cuando el tamaño del bloque sintáctico es demasiado pequeño), además de su posición absoluta (`starts` y `ends`) relativa al contenedor, cuyo contenido también se incluye como contexto de la acción en caso de existir. En el Apartado 4.3.2 se dará un ejemplo ilustrativo sobre la localización de contexto y la primitiva final generada, a partir de la eliminación de un literal en una página web.

Todas las acciones necesitan de un contexto de localización, por lo tanto este módulo es aplicado como base a otro tipo de acciones más complejas, como por ejemplo la creación de objetos de presentación, que necesitarán la información de este módulo para poder abstraer más datos sobre lo que el usuario desea hacer en todo momento.

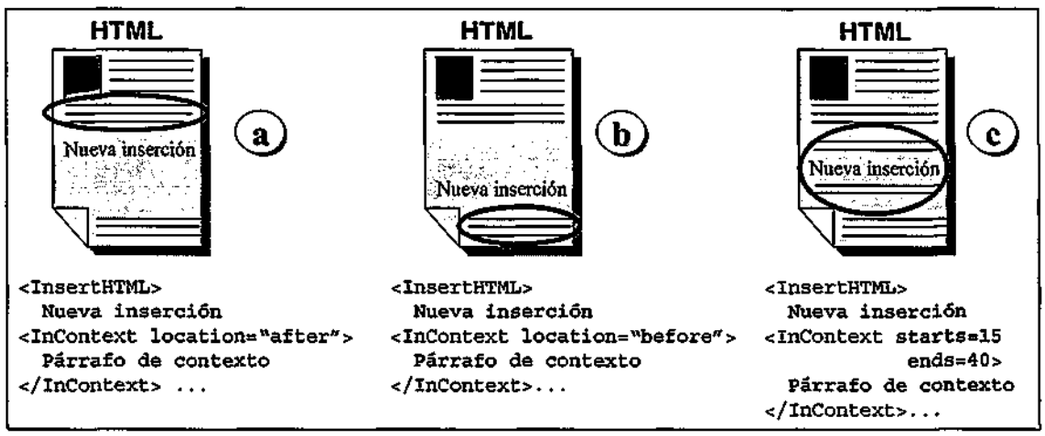


Figura 4.4: Localización del contexto de la acción del usuario

b) Módulo de identificación de estructuras especiales.

Este módulo se encarga de identificar las estructuras, objetos o widgets HTML (tablas y listas), sobre las que se ha producido una acción del usuario. Este módulo sabe abstraer características de dichas estructuras, como por ejemplo el número y tipo de los elementos que la forman, así como la geometría del propio widget. En la Figura 4.5 se describen cuatro casos distintos de acciones sobre estructuras especiales, en concreto dos casos de creación de widgets (`CreateWidget`), uno de inserción (`InsertHTML`)

y otro de borrado de widgets (`DeleteWidget`). En todos ellos el módulo abstrae información relativa al widget en cada caso, como el número de filas y columnas (`cols`, `rows`) cuando se trata de tablas, o el número de elementos (`items`) en listas. Como se refleja en la Figura 4.5, cada uno de estos objetos de la presentación es etiquetado y se le asigna un identificador único (`id`) y un tipo, dependiendo de si es una tabla (`type="Table"`) o una lista (`type="List"`). De esta forma serán identificados posteriormente en el servidor a la hora de proceder a realizar cambios sobre estos widgets, como por ejemplo en la inserción o eliminación de elementos.

Como se puede comprobar en la Figura 4.5, la información relativa al contexto sigue siendo necesaria en este caso, pues al crear uno u otro widget es necesario identificar en qué lugar del documento serán insertados. En el caso de producirse acciones en el interior del widget, como la inserción de código HTML en la celda de una tabla ya creada (Figura 4.5), la información de contexto se referirá ahora al lugar donde debe producirse la acción; la fila (`row_n`) y la columna (`col_n`), e incluso el lugar dentro del párrafo, en el caso de que ya existiera HTML dentro de dicha celda, en el que se producirá la inserción del nuevo HTML. Este mecanismo también es válido para la creación de widgets anidados (p.e. una tabla dentro de otra). En el caso de borrado de un widget, tal y como muestra la Figura 4.5, solamente es necesario reconocer el identificador (`id`) del widget en cuestión, dado que dicho widget ya ha sido registrado como elemento de la presentación en el paso anterior de creación, siendo la herramienta la encargada de localizar dicho widget a partir de su código único.

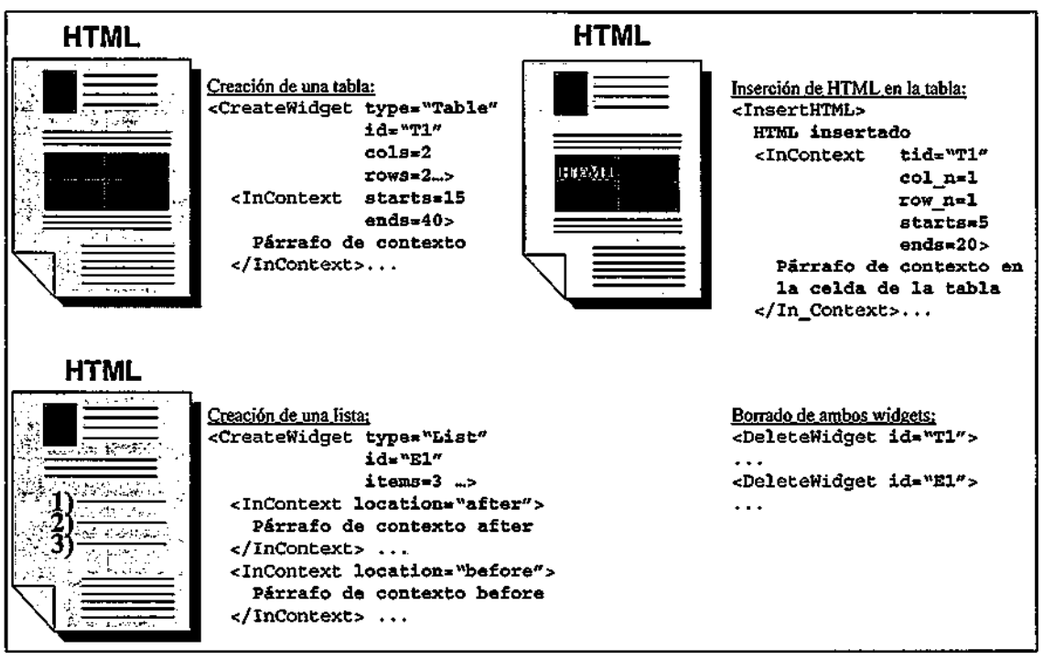


Figura 4.5: Identificación de controles HTML

c) Módulo de creación de primitivas.

Este módulo toma las acciones del usuario junto con el contexto sintáctico encontrado en los módulos anteriores, construyendo primitivas como las mostradas en la Figura 4.5, las cuales finalmente constituirán el modelo de monitorización en XML. Adicionalmente, este módulo también se encarga de analizar las correspondencias entre las acciones del usuario y las primitivas generadas, catalogando éstas en diferentes grupos según la acción acometida.

4.3.2 El modelo de monitorización

Las primitivas que forman el modelo de monitorización son elaboradas a partir de la catalogación de las acciones llevadas a cabo por el usuario. En realidad no existe una primitiva por cada acción del usuario, en su lugar las acciones son analizadas y agrupadas en categorías más generales. La explicación de esto es que a más de una acción le puede corresponder una misma primitiva, por ejemplo copiar un fragmento HTML o insertarlo desde teclado son dos acciones distintas que son abstraídas bajo una misma primitiva: `InsertHTML`, lo mismo sucede con las acciones de cortar y borrar (`DeleteHTML`). Esto, sin embargo, no ocurre para otras primitivas de copia y pegado que afectan a widgets HTML donde, como se verá más adelante, es necesario saber el origen y el destino de la copia para poder inferir el flujo de la información y poder realizar transformaciones automáticas, mediante un agente especializado, entre este tipo de estructuras. En cualquier caso, la codificación de las primitivas no es estática, sino que depende del contexto de la acción en sí, siendo DESK capaz diferenciar una primitiva de inserción de HTML en una tabla de otra que inserta HTML dentro de un párrafo de un elemento de una lista. De esta forma el contexto variará reflejando, en el primer caso, la localización dentro de la tabla, y en el segundo la posición dentro del párrafo del elemento de la lista afectado. Así pues DESK considera inicialmente, aunque la lista es ampliable, las siguientes primitivas:

```
<ApplyStyle    type=style...>    <RemoveHTML>
<DeleteStyle   type=style...>    <PasteFragment  from=WID1
<CreateWidget  type=WType                        to=WID2...>
               id=WID...>        <CopyFragment   from=WID1
<DeleteWidget  type=WType                        to=WID2...>
               id=WID...>        <ChangeWidget   from=WID1
<InsertHTML>                                     to=WID2>
```

Desde un primer momento se decidió que el número de primitivas no fuera demasiado grande, reduciendo así el número de primitivas del lenguaje que compone el modelo y aumentando la generalidad en la codificación de las acciones llevadas a cabo por el usuario. En general, cada una de las primitivas tiene sus propios parámetros y atributos, y estos son identificados en el servidor a partir de la información que cada primitiva codifica. La ventaja de esta filosofía es que el conjunto es en gran medida escalable, y no sería demasiado complejo añadir nuevas primitivas que reflejasen nuevas acciones a codificar. La estructura general de una primitiva es la siguiente:

```
<Nombre_Primitiva ID atributos_específicos>
  <TextAffected>
    Texto afectado por el cambio del usuario
  </TextAffected>
  <InContext starts ends after before>
    Localización con respecto al bloque sintáctico de
    referencia
  </InContext>
</Nombre_Primitiva>
```

El modelo de monitorización se compone de una secuencia ordenada respecto al tiempo de este tipo de primitivas, de forma que a cada primitiva se le asigna un ID en función de su orden secuencial de ejecución. Como ya se explicó anteriormente, `InContext` refleja el contexto o el bloque sintáctico de referencia que se utilizará para localizar posteriormente el cambio en los modelos subyacentes, además `TextAffected` refleja la información que ha sido objeto del cambio.

Por ejemplo, en un documento web sobre el algoritmo de Dijkstra como el de la Figura 4.6, eliminar la palabra "Dijkstra" del literal "El Algoritmo de Dijkstra." provocaría la siguiente entrada en el modelo de monitorización:

```
<DeleteHTML ID="1">
  <TextAffected> Dijkstra </TextAffected>).
  <InContext starts=16 ends=24 before="Procedimiento">
    El Algoritmo de Dijkstra.
  </InContext>
</DeleteHTML>
```

En la Figura 4.6 se refleja este ejemplo bajo DESK, donde el bloque sintáctico es la cadena "El Algoritmo de Dijkstra." y la posición donde comienza el texto a eliminar (*Dijkstra*) es justo la número 16 (`starts=16`) dentro del bloque sintáctico, acabando el último carácter de la palabra a eliminar en la posición 24 (`ends=24`). Además, el atributo `before` de la cláusula `InContext` señala que el bloque sintáctico se encuentra justo antes del párrafo "Procedimiento", correspondiente a otro objeto del dominio distinto. La primitiva descrita anteriormente se generaría justo al eliminar el fragmento "Dijkstra", quedando una instantánea como la de la derecha, según la Figura 4.6.

Como ejemplo real de ejecución, a continuación se muestra un fragmento de modelo de monitorización compuesto por cuatro primitivas. La primera de ellas (`ID="1"`) es el resultado de eliminar (cortar) el código del algoritmo de Dijkstra para posteriormente (primitiva `ID="2"`) insertarlo (pegarlo) en otra posición distinta del documento, como puede verse por el contexto asociado a la segunda primitiva. Posteriormente, en la primitiva número tres (`ID="3"`) se refleja la creación de una lista de elementos, ordenada y comenzando en 1, a partir el código del algoritmo de Dijkstra que se manipuló en la primitiva anterior. Finalmente, en la siguiente primitiva (`ID="4"`) se procede a cambiar el estilo de la palabra "Algoritmo" que se encuentra

dentro del bloque sintáctico "El titular del algoritmo es:" sujeto de la acción del usuario, es decir, el cambio consiste en subrayar dicha palabra. Normalmente, siempre que se mantenga la secuencialidad de las acciones del usuario sobre un contexto cercano, no será necesario identificar los párrafos de alrededor del bloque sintáctico mediante atributos `after` y `before`, esto lo controla la propia herramienta mediante el módulo de localización de contexto y caracterización del bloque sintáctico foco de la acción del usuario en la edición del documento.

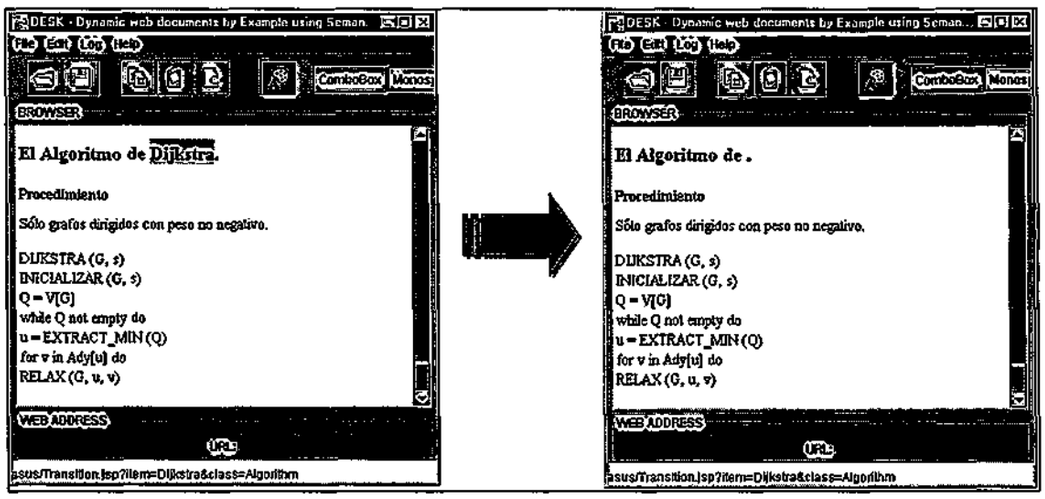


Figura 4.6: Eliminación de un fragmento de HTML en DESK

```
<?xml version="1.0" encoding="UTF-8"?>
<MonitoringModel>
 <RemoveHTML ID="1">
  <TextAffected> <p> DIJKSTRA (G, s)<br>INICIALIZAR (G, s)<br>Q =
V[G]<br>while Q not empty do <br>
u = EXTRACT_MIN (Q)<br>for v in Ady[u] do<br>RELAX (G, u, v)
    </p> </TextAffected>
  <InContext starts="0" ends="120"> DIJKSTRA (G, s) INICIALIZAR (G, s)
Q = V[G] while Q not empty do u = EXTRACT_MIN (Q) for v in Ady[u] do
RELAX (G, u, v)</InContext>
 </RemoveHTML >
 <InsertHTML ID="2">
  <TextAffected> <p> DIJKSTRA (G, s)<br>INICIALIZAR (G, s)<br>Q =
V[G]<br>while Q not empty do<br>
u = EXTRACT_MIN (Q)<br>for v in Ady[u] do<br>RELAX (G, u, v)
    </p> </TextAffected>
  <InContext starts="12" ends="172"> Solo grafos dirigidos con peso no
negativo.</InContext>
 </InsertHTML >
<CreateList ID="3" nature="sorted" type="1" start="1">
  <TextAffected>DIJKSTRA (G, s) INICIALIZAR (G, s) Q = V[G] while Q
not empty do u = EXTRACT_MIN (Q) for v in Ady[u] do RELAX (G, u,
v)</TextAffected>
  <InContext>DIJKSTRA (G, s) INICIALIZAR (G, s) Q = V[G] while Q not
empty do u = EXTRACT_MIN (Q) for v in Ady[u] do RELAX (G, u,
v)</InContext>
  <ListItems>
   <Item-1> DIJKSTRA (G, s)      </Item-1>
   <Item-2> INICIALIZAR (G, s)   </Item-2>
```

```
    <Item-3> Q = V[G]                <Item-3>
    <Item-4> while Q not empty do <Item-4>
    <Item-5> u = EXTRACT_MIN (Q)   <Item-5>
    <Item-6> for v in Ady[u] do    <Item-6>
    <Item-7> RELAX (G, u, v)       <Item-7>
   </ListItems>
  </CrateList>
  <ApplyStyle ID="4" type="UNDERLINE">
   <TextAffected>algoritmo</Text_Affected>
   <inContext start_text="15" end_text="24">El titular del algoritmo
es:</InContext>
  </ApplyStyle>
</MonitoringModel>
```

4.4 DESK servidor

La aplicación DESK del servidor se encarga de recibir el modelo de monitorización, enriquecerlo con la semántica extraída de los modelos subyacentes de PEGASUS y proceder a efectuar los cambios uno por uno, según están codificados en el propio modelo de monitorización. Finalmente, y como resultado del proceso, DESK servidor envía información HTML sobre lo acontecido durante el proceso de inferencia y modificación.

La forma de inferir los cambios en el servidor se lleva a cabo mediante la utilización de heurísticas que permiten caracterizar cada uno de los cambios acometidos por el cliente en la herramienta. De esta forma, DESK servidor caracteriza los objetos relacionados en dichos cambios, encuentra relaciones entre objetos del dominio y de la presentación, y lleva a cabo modificaciones en estos modelos que permitirán alterar el procedimiento por el cual se generarán las futuras páginas web.

DESK servidor también detecta posibles ambigüedades en las modificaciones del usuario, y al final del proceso envía información a la aplicación cliente sobre lo acontecido durante el proceso de inferencia, de esta forma el usuario tiene constancia de si se ha producido alguna anomalía o alguna ambigüedad que necesita ser resuelta a través de la interacción directa con el usuario.

4.4.1 Mecanismo de inferencia en el servidor

Para llevar a cabo su labor, DESK servidor sigue una serie de pasos reflejados en la Figura 4.7, cuyo significado se detalla a continuación:

1) Una vez que DESK cliente ha enviado el modelo de monitorización, DESK servidor comienza a analizar cada una de las primitivas que componen el modelo con el objetivo de encontrar un contexto semántico para cada primitiva, aplicando *heurísticas de alto nivel* (H_H) que indagan en el modelo del dominio y en el de la presentación para aportar información contextual más completa y precisa sobre los cambios. Estas primitivas comprenden una serie de módulos especializados en encontrar información sintáctica, así como relaciones entre los distintos elementos del modelo del dominio y de la presentación. Como se explicará más adelante, la

misión principal de estos algoritmos es la caracterización de objetos principales y de relaciones semánticas, así como atributos y fragmentos del conocimiento que intervienen en los cambios acometidos por el usuario.

2) La información semántica encontrada es añadida a cada una de estas primitivas, dando como resultado un modelo de la monitorización enriquecido.

3) Este modelo enriquecido será utilizado finalmente por el módulo de gestión de cambios para proceder a realizar sobre PEGASUS las modificaciones que representan los cambios que el usuario hizo bajo la herramienta de edición en el cliente. De esta forma, y como resultado final del proceso, se consigue una generalización de los cambios, en los modelos subyacentes, a partir de la modificación inicial de una página concreta por parte del usuario.

Los cambios a realizar pueden afectar tanto al modelo del dominio, si se añade o modifica conocimiento del dominio, como al modelo de la presentación, si el cambio afecta directamente a los objetos de la presentación o al código JSP/HTML presente en la plantilla de presentación de PEGASUS. Antes de realizar los cambios se indaga sobre si el usuario que interactúa con la herramienta está autorizado para realizar dichos cambios. Si es así, el gestor de cambios actualiza directamente los modelos internos de PEGASUS, dando como resultado páginas de información para el usuario así como el propio documento web ya actualizado. En adelante, cuando se visualicen objetos de las mismas clases, la información aparecerá cambiada según las modificaciones que el usuario especificó.

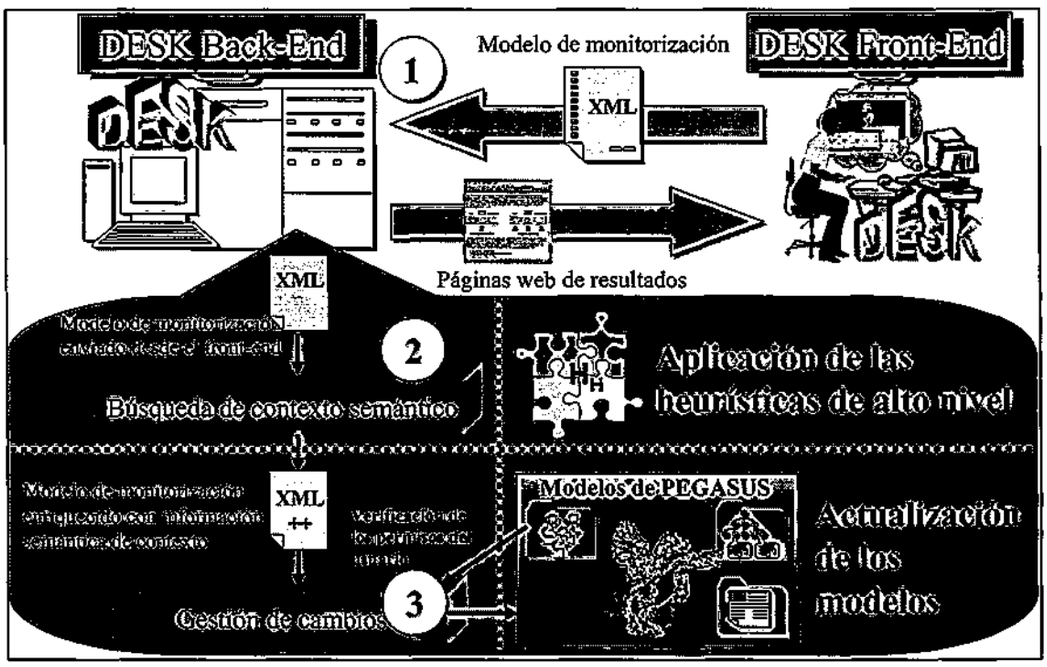


Figura 4.7: DESK servidor

4.4.2 Heurísticas de alto nivel

El principal cometido de las heurísticas de alto nivel es la de proporcionar una serie de algoritmos utilizados para localizar contexto semántico, en base al contexto sintáctico proporcionado por las heurísticas de bajo nivel. La Figura 4.8 es una visión detallada del módulo de aplicación de heurísticas de alto nivel de la Figura 4.7, donde se ilustran los distintos módulos que forman estas heurísticas. Según se detalla en la Figura 4.8, el eje central es un módulo principal encargado de la localización de contexto semántico a partir de información del dominio, así como de una abstracción del modelo de la presentación generada mediante otro módulo adicional. Por otro lado, el módulo de desambiguación hace frente a la posibilidad de tener distintos caminos donde elegir, intentando resolver la ambigüedad mediante conocimiento sintáctico y del dominio. Adicionalmente, existe un módulo encargado de construir una abstracción del modelo de la presentación, cuya información será utilizada por el módulo de localización de contexto semántico para una búsqueda y caracterización de objetos más eficiente.

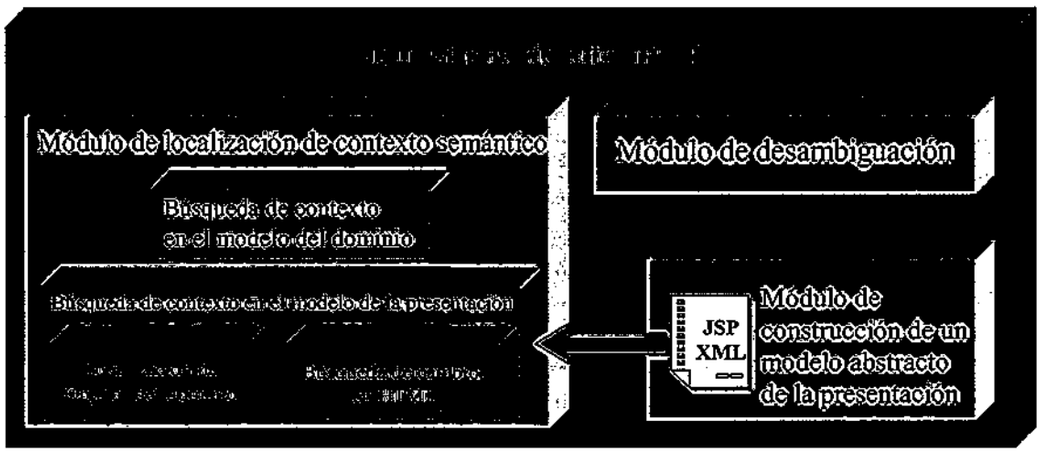


Figura 4.8: Heurísticas de alto nivel

4.4.2.1 Módulo de localización de contexto semántico

Este es el módulo principal y su cometido es el de analizar las primitivas para encontrar la correspondiente semántica asociada a cada una, obteniendo esta información a partir de los otros submódulos, y ordenando y procesando posteriormente esta información para encontrar y corregir posibles ambigüedades. El significado de esta fase de búsqueda y análisis tiene sentido si se parte del hecho de que lo que se hace es recorrer un camino inverso, partiendo de cambios realizados por el usuario sobre el código HTML, para acabar infiriendo vínculos en el dominio sobre los objetos involucrados en dichos cambios.

Como partes integrantes del módulo de localización semántica, se encuentran distintos módulos o algoritmos encargados de tareas bien definidas, como la localización de objetos del dominio, la gestión de cambios en bloques HTML y, en

general, la búsqueda de contexto semántico tanto en el modelo de la presentación como en el modelo del dominio de PEGASUS.

a) Búsqueda de contexto en el modelo del dominio

Para desarrollar la búsqueda de contexto en el modelo del dominio se utilizan algoritmos especializados que recorren la ontología del dominio junto con sus instancias para localizar posibles candidatos a ser el contexto semántico de la primitiva analizada. La información que busca este módulo viene dada en términos de objetos del dominio junto con atributos y fragmentos atómicos. El valor de vuelta es la ruta (path) a dichos objetos en el modelo del dominio desde el *objeto principal*. Este módulo también es capaz de detectar relaciones entre objetos, para acceder a la información de uno a partir del otro en el caso de que el usuario haya decidido realizar un cambio sobre una relación multivaluada o no. Como resultado final del proceso se añade a la correspondiente primitiva del modelo de monitorización una nueva cláusula `FoundIn` que refleja información de alto nivel sobre la semántica asociada a dicha acción.

En la Figura 4.9 aparecen tres ejemplos, donde para cada uno de ellos se supone una entrada en el modelo de monitorización con su correspondiente contexto sintáctico. Utilizando esta información se ha localizado, para cada uno de los casos, información semántica relativa al cambio. En el caso número 1 el cambio se ha localizado en un atributo (`title`) perteneciente a un objeto del modelo del dominio (`Dijkstra`), instanciado a partir de una clase de la ontología del dominio (`Algorithm`). En el caso número 2, el cambio se ha detectado sobre un atributo (`title`) de una relación multivaluada entre objetos de la red semántica del dominio (`procedure`). Y en el caso número 3 el cambio se localiza dentro de un fragmento atómico (`AtomicFragment`), el cual representa texto HTML perteneciente a la red semántica del dominio. Como puede comprobarse en los ejemplos, esta información se almacena en forma de atributos dentro de la cláusula `FoundIn` añadida para enriquecer semánticamente cada una de las primitivas que forman el modelo de monitorización. El objetivo final es inferir una caracterización de un fragmento modificado como parte integrante del modelo del dominio. Por poner un ejemplo, en el modelo del dominio hay unidades como el algoritmo de Dijkstra, un procedimiento, etc., y de lo que se trata es de, una vez encontrado el ítem, saber cómo caracterizarlo, es decir, de saber si es un título del objeto `Dijkstra`, un atributo del procedimiento o un fragmento de HTML. De esta forma, y siguiendo con el ejemplo, para el caso número 2 de la Figura 4.9 se llega a inferir que lo que se ha manipulado es el título del procedimiento del algoritmo de Dijkstra.

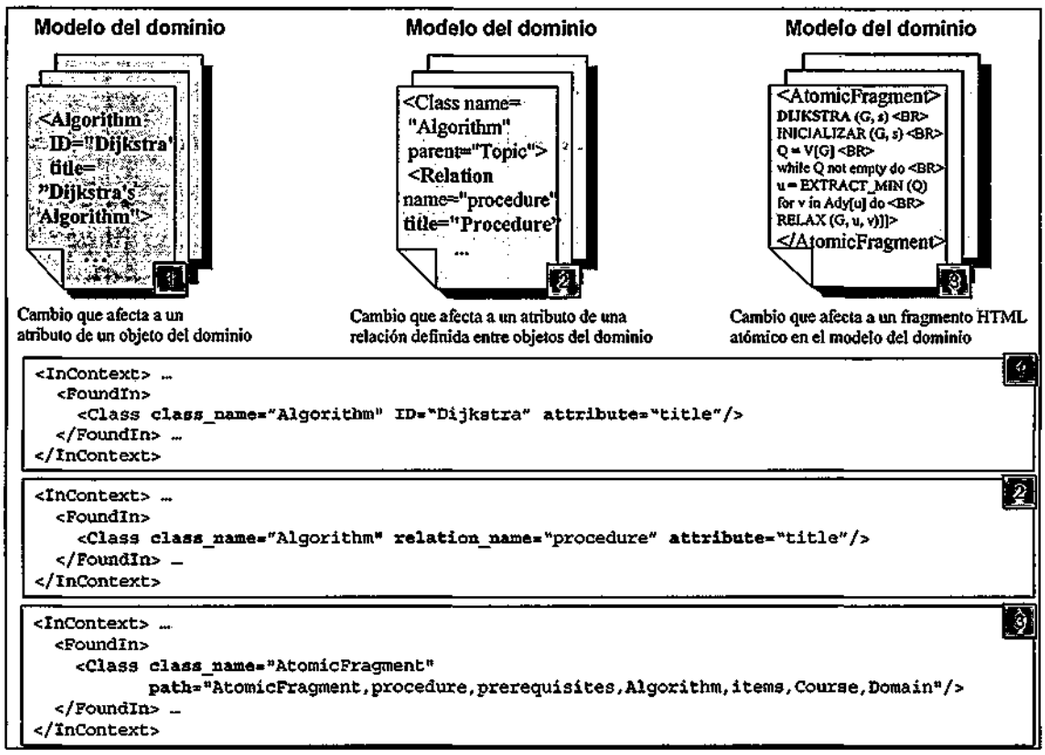


Figura 4.9: Distintos ejemplos de contexto semántico

Esta misma filosofía puede aplicarse a otro tipo de directivas, representadas dentro de la plantilla de presentación como relaciones multivaluadas entre objetos del dominio. En la Figura 4.10 se muestra un ejemplo de detección de cambios en una lista de elementos generada a partir de la relación multivaluada `prerequisites` de la plantilla de la presentación. En este tipo de directivas el proceso de detección es más complejo que el procedimiento descrito anteriormente, pues hay que localizar valores no directos como el nombre de la relación y el camino de acceso hasta el *objeto principal* que define la relación multivaluada.

La obtención del *objeto principal* es un paso fundamental para la caracterización de los cambios. De lo que se trata en este caso es de poder localizar cuál es el objeto fuente o raíz que soporta las relaciones entre los objetos que han sido localizados o caracterizados como partes fundamentales de los cambios realizados por el usuario. Esto servirá para dar una ruta de acceso completa, obteniendo así una semántica que permita una mejor identificación de los cambios.

Una forma sencilla de obtener el objeto principal es mediante la URL o dirección web utilizada para solicitar a PEGASUS una página web. Estas direcciones son siempre del tipo:

```
http://astreo.ii.uam.es:8081/PEGASUS/Transition.jsp?item=Di
jkstra&class=Algorithm
```

De esta forma se obtiene la dirección al servidor PEGASUS, pero también el nombre del recurso que se está solicitando, en este caso el objeto *Dijkstra* (`item=Dijkstra`) de la clase *Algorithm* (`class=Algorithm`). Es decir, en ese momento el objeto principal que está siendo procesado es el objeto *Dijkstra*, y se tomará como referencia en el servidor, en el momento de la generación de la página HTML, como valor semántico de utilidad.

Como ejemplo ilustrativo de este mecanismo, representado en la Figura 4.10, supongamos que el usuario quiere efectuar un cambio en el literal `Item 1`, resultado de una relación multivaluada llamada `prerequisites`. El algoritmo opera buscando en el modelo del dominio el valor modificado, en este caso codificado bajo el atributo `title` del objeto `ALG1`. Como se mencionó anteriormente, el objeto principal se obtiene a partir de la URL de acceso al propio objeto. De esta forma hay que subir por el árbol subyacente en la topología de la red semántica hasta alcanzar la raíz o bien el objeto principal. Si se ha llegado a la raíz sin encontrar dicho objeto, hay que repetir de nuevo la búsqueda en anchura para cada elemento del árbol hasta dar con el objeto principal `Dijkstra`, el cual contiene la relación `prerequisites` que a su vez contiene el objeto `ALG1`. De esta forma, DESK guarda toda la información resultante en el recorrido, para así inferir finalmente que `Item 1` es un atributo del objeto `ALG1` que ha sido renderizado a partir de la relación multivaluada `prerequisites` del objeto principal `Dijkstra`, perteneciente a la clase `Algorithm` que además se corresponde con el nombre de la plantilla. De esta forma, se termina caracterizando el elemento mediante la expresión de acceso "el título del prerrequisito del algoritmo de Dijkstra". Utilizando esta filosofía es relativamente sencillo controlar los elementos generados a partir de relaciones entre objetos.

De forma similar, si el usuario decide desde DESK cambiar la distribución de los datos (de vertical a horizontal), DESK tratará de identificar exactamente la procedencia de los datos utilizando el mecanismo descrito anteriormente, localizando la plantilla de la presentación adecuada donde procederá a cambiar el valor de la expresión:

```
<%=prerequisites.tree("vertical")%>
```

por el de:

```
<%=prerequisites.tree("horizontal")%>.
```

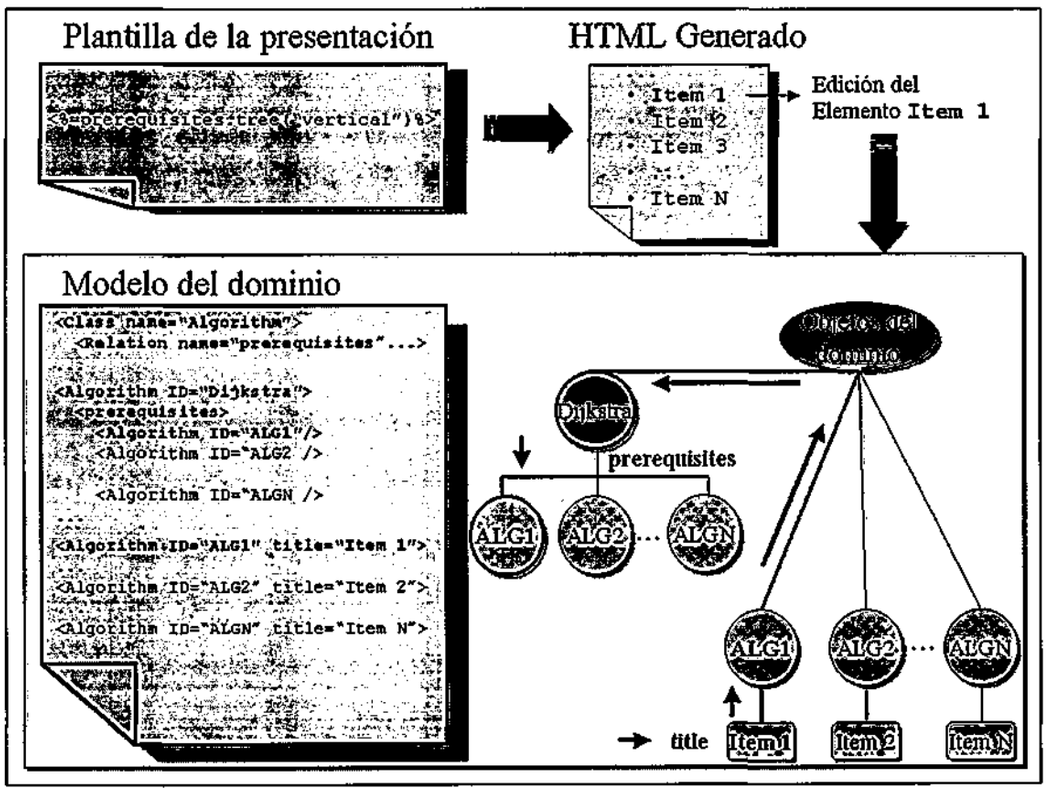


Figura 4.10: Localización de objetos en relaciones multivaluadas a partir de la topología de la red semántica

b) Búsqueda de contexto en el modelo de la presentación

Al igual que existe un módulo encargado de encontrar contexto exclusivamente en el modelo del dominio, es necesario también tener un módulo encargado de caracterizar el lugar de aparición de los objetos dentro del modelo de la presentación. Este conjunto de algoritmos realizan distintos tipos de búsquedas sobre el modelo o la plantilla de la presentación, en el caso de que el cambio afecte a este modelo en vez de afectar al modelo del dominio. En este caso se trata de localizar los objetos del dominio presentes en modelo de la presentación, o bien los fragmentos HTML/JSP, los cuales son objeto de un cambio realizado por el usuario. Para realizar estás búsquedas, este módulo se ayuda de un modelo abstracto de la presentación, que se describirá más adelante, creado en tiempo real para localizar la información mucho más rápidamente, analizando y diseccionando cada uno de los componentes de la presentación (Figura 4.11), así como la relación entre los distintos objetos (Figura 4.12) (análisis de variables de presentación, etiquetas que varían la renderización de los objetos del dominio, definición de objetos del dominio en la plantilla y representación de relaciones multivaluadas que afectan a más de un objeto).

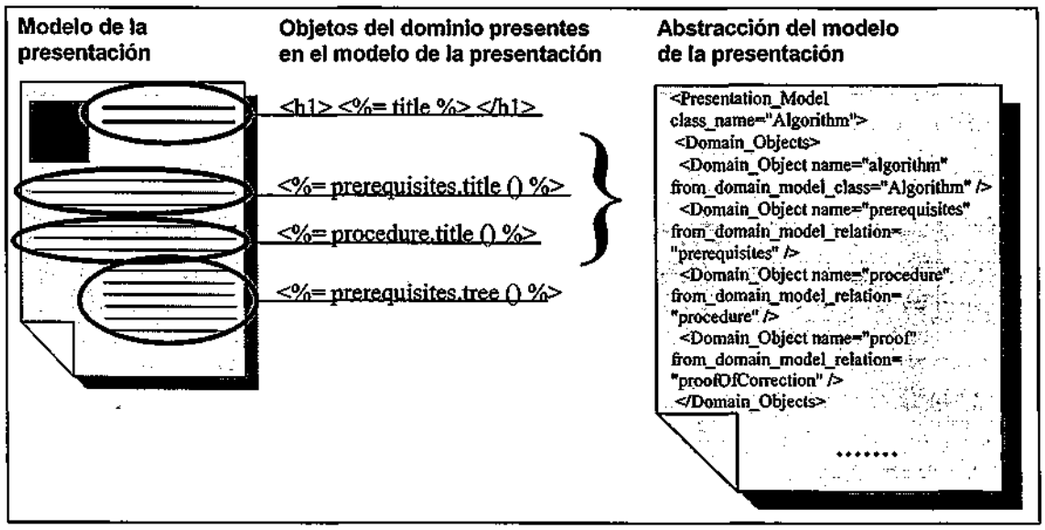


Figura 4.11: Disección de la presentación en objetos utilizando un modelo abstracto de la presentación

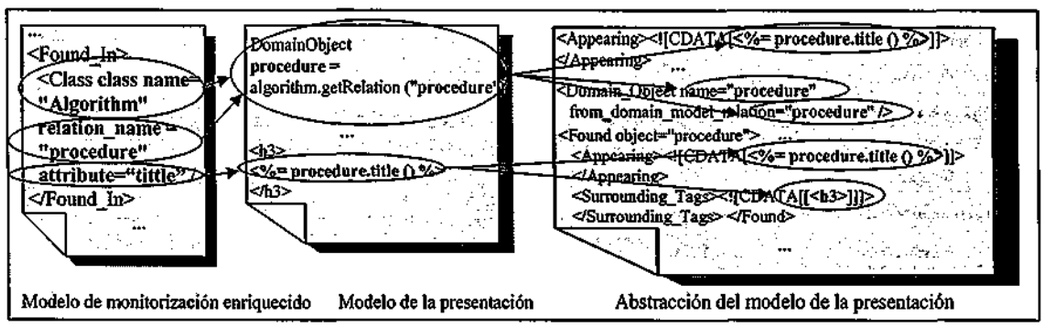


Figura 4.12: Correspondencias entre los distintos modelos para el módulo de reconocimiento de objetos del dominio en la presentación

4.4.2.2 Módulo de construcción de un modelo abstracto de la presentación.

Este módulo crea una representación abstracta de la plantilla de la presentación, de forma que posteriormente el módulo de búsqueda semántica de contexto en el modelo de la presentación pueda localizar información sobre los objetos que le interesan, y que han sido caracterizados por estar relacionados con cambios que ha efectuado el usuario sobre la presentación.

Básicamente, esta modelización ad-hoc se lleva a cabo a través de la detección de partes integrantes de la presentación, detectando los objetos del dominio, incluso si están definidos como variables de presentación, así como el paso de mensajes entre objetos del dominio junto con las etiquetas subyacentes a dichos objetos.

Como puede verse en las Figuras 4.11 y 4.12, este módulo crea una representación útil del modelo de la presentación basada en la caracterización de las partes integrantes del modelo del dominio en la plantilla de la presentación. Para cada una o varias de las entradas de esta abstracción de la presentación existe una

correspondencia en los modelos del dominio y de la presentación, pues al fin y al cabo se trata de codificar, de una forma más *computable*, la información que se almacena en la plantilla para ser accedida más fácilmente y poder localizar lo objetos de forma automática en el modelo del dominio. La información que se codifica en este modelo abstracto es la siguiente:

- Objetos del dominio: Se definen mediante la cláusula `<DomainObjects>`, donde por cada objeto definido habrá una entrada del tipo `<DomainObject>`, donde se codifica la relación en la que interviene y la clase de pertenencia.
- Variables de plantilla y atributos de objetos: Mediante la cláusula `<AnotherObjects>` se crea una entrada `<AnotherObject>` donde se recogen definiciones de variables o atributos de objetos del dominio en la plantilla de la presentación, codificando además el tipo del atributo así como el objeto y la clase de pertenencia.
- Aparición de objetos en la plantilla: Se codifican dentro de la cláusula `<AppearingDomainObjects>`, donde por cada objeto podemos encontrar una entrada del tipo `<Found>` con el nombre del objeto tal y como aparece en la presentación, junto con los métodos invocados en cada caso (`<Appearing>`) y las etiquetas HTML que lo rodean (`<SourroundingTags>`).

El sistema crea en tiempo real un modelo abstracto de este tipo para cada plantilla de presentación PEGASUS involucrada en un cambio del usuario. Un ejemplo real de esta abstracción puede verse a continuación, donde podemos ver definidos, para la plantilla `Algorithm` (`<PresentationModel class_name="Algorithm">`), distintos objetos y relaciones, como una instancia de la propia clase `Algorithm`, relaciones multivaluadas como `prerequisites`, `procedure`, y `proof`, además del atributo `title` que codifica el título del algoritmo para esta plantilla. Finalmente se definen las apariciones de estos objetos junto con el formato de su llamada (invocación de métodos o paso de mensajes, p.e. `procedure.title()`), además de las etiquetas HTML que rodean a la llamada en cada caso.

```
<?xml version="1.0" encoding="UTF-8"?>
<PresentationModel class_name="Algorithm">
 <DomainObjects>
  <DomainObject name="algorithm" from_domain_model_class="Algorithm"
/>
  <DomainObject name="prerequisites"
from_domain_model_relation="prerequisites" />
  <DomainObject name="procedure"
from_domain_model_relation="procedure" />
  <DomainObject name="proof"
from_domain_model_relation="proofOfCorrection" />
  </DomainObjects>
  <AnotherObjects>
  <AnotherObject name="title" type="String" from_class="Algorithm"
from_attribute="TITLE" />
```

```
</AnotherObjects>
< AppearingDomainObjects >
 <Found object="prerequisites">
  <Appearing> <%= prerequisites.title (0) %> </Appearing>
  <Surrounding_Tags> <h6> </SurroundingTags>  </Found>
 <Found object="procedure">
  <Appearing> <%= procedure %> </Appearing>  </Found>
 <Found object="procedure">
  <Appearing> <%= procedure.title (0) %> </Appearing>
  <Surrounding_Tags> <h6> </SurroundingTags>  </Found>
 <Found object="proof">
  ◇◇◇◇
</PresentationModel>
```

4.4.2.3 Módulo de desambiguación.

Este módulo controla los posibles casos de ambigüedad que puedan surgir durante el proceso de inferencia en el servidor. En DESK se entiende por ambigüedad las primitivas del modelo de monitorización que poseen más de un contexto posible, es decir, aquéllas en las que los módulos de localización semántica han encontrado distintos lugares, en el modelo del dominio y de la presentación, donde este cambio puede verse afectado.

La ambigüedad surge del hecho de que DESK intenta seguir el camino inverso al de PEGASUS, donde partiendo de una página HTML se intenta llegar a los modelos constructores que han dado lugar a dicha página, existiendo la posibilidad de una ambigüedad inherente en el proceso, ya que existe más de una manera de generar una misma página. Pueden darse casos de ambigüedad cuando un valor atómico se repite en varios atributos de la base de conocimiento. También cuando existen varias formas posibles de referenciar a un elemento de la base de conocimiento (diferentes relaciones). DESK intenta solventar este problema utilizando heurísticas de desambiguación.

El módulo de desambiguación trata de averiguar, para aquellas primitivas que tienen más de un contexto semántico (varias entradas bajo la cláusula `FoundIn` en el modelo de monitorización enriquecido), cual es el contexto real, observando el entorno de aparición del fragmento HTML o del objeto de la presentación. La forma de realizar este proceso es mediante la comparación del bloque sintáctico y su entorno (codificado con los atributos `after` y `before`) con la información semántica generada. Tras estudiar cada una de estas apariciones, este módulo compara los resultados de la búsqueda del entorno cercano de aparición y clasifica los resultados, decidiendo, en función de la similitud, cuál es el entorno ganador (más parecido) de los que se codifican dentro de la cláusula `FoundIn`.

La justificación del uso de este mecanismo viene dada por el hecho de que lo que intenta seguir es un camino inverso, partiendo de una página HTML para acabar caracterizando objetos del dominio involucrados en el cambio. Es por tanto factible el hecho de que un mismo cambio pueda verse reflejado en varios de los elementos del modelo del dominio (p.e. coincidencia en dos atributos de distintos objetos, relaciones o

incluso fragmentos atómicos HTML), teniendo que diferenciar la herramienta cuál es el contexto más apropiado a partir del mecanismo descrito anteriormente.

Los casos de ambigüedad que pueden aparecer en la inferencia, y que DESK trata de resolver con información sobre del dominio, son los siguientes:

- Ambigüedad producida al encontrar una misma aparición en distintos elementos del modelo del dominio. Este es el caso en el que un mismo literal, como por ejemplo "El Algoritmo de Dijkstra", puede aparecer como atributo de un objeto, como nombre de una relación, o como HTML en un fragmento atómico. En este caso, DESK utiliza información sintáctica del contexto para averiguar los objetos circundantes, eligiendo de entre todas las posibilidades la más parecida. Para averiguar la relación entre el contexto sintáctico que rodea a un caso ambiguo y su relación de alto nivel con los objetos del dominio, se utilizan nuevamente las heurísticas de alto nivel, que como ya se vio analizan el modelo del dominio para extraer información sobre la red semántica de objetos que lo componen.
- Ambigüedad producida al encontrar una misma aparición en distintos elementos del modelo de la presentación. En este caso DESK utiliza la información del modelo abstracto de la presentación, donde encuentra información sobre los bloques constructivos de la plantilla y su relación con el modelo del dominio. Nuevamente se utiliza información de contexto, extraída del modelo abstracto de la presentación, para elegir la mejor opción entre las posibles que puedan aparecer.
- Ambigüedad producida al encontrar una misma aparición, tanto en el modelo de monitorización como en el modelo de la presentación. Este caso surge cuando tenemos, por ejemplo, un literal en la plantilla de la presentación que coincide con un elemento del modelo del dominio. Un ejemplo de este caso puede ser el hecho de que el literal "El Algoritmo de Dijkstra" aparezca como código HTML dentro de la plantilla de la presentación, pero que también se corresponda con un atributo del modelo del dominio. En éste caso DESK vuelve a utilizar nuevamente información contextual para, en esta ocasión, discernir entre ubicar el cambio en el modelo de la presentación o del dominio. Como ayuda, DESK utiliza además, de forma simultánea, información del dominio y del modelo abstracto de la presentación, para poder llegar a un conjunto reducido de apariciones del cual se extraerá la aparición más acertada.
- Casos de ambigüedad irresoluble. Es el caso en el cual DESK no puede elegir una aparición apropiada dentro del conjunto de posibles apariciones para un solo cambio realizado por el usuario. Esto se produce cuando más de un elemento tiene la misma probabilidad, por *matching de contexto*, de ser candidato para resolver la ambigüedad. En este caso DESK solicita la ayuda del usuario, presentando en una página de resultados los casos de ambigüedad encontrados junto con su contexto, para que sea el propio usuario el que elija el candidato apropiado para resolver la ambigüedad.

Para paliar la ambigüedad que se produce en el proceso de inferencia, el módulo de desambiguación de DESK utiliza información disponible en el dominio para la caracterización y localización de objetos de la presentación. De esta forma, la información encontrada se procesa mediante *mathing* de contexto, que permite comparar cual es el contexto más acertado para resolver la ambigüedad.

En la Figura 4.13 tenemos un ejemplo de un caso en el cual, dentro de una misma primitiva del modelo de monitorización, aparecen dos posibles contextos semánticos (doble entrada en <FoundIn>) para el cambio realizado por el usuario. DESK estudia la información disponible para estos dos casos, uno encontrado en el atributo (`title`) de un objeto (`ID="Dijkstra"`) y otro encontrado dentro de un fragmento atómico, junto con la información existente en la primitiva sobre el contexto sintáctico calculado por las heurísticas de bajo nivel. Después de evaluar los casos existentes, el sistema llega a la conclusión de que el primero de ellos es el más apropiado, pues los atributos `after` y `before`, en este caso, discriminan perfectamente la localización del texto modificado (antes de "Algoritmo de Relajación" y después del texto "Solo gráfos dirigidos...").

Aunque en algunos casos el disponer de toda esta información puede resultar redundante, la idea es tener un registro de todos los lugares donde aparece una ocurrencia similar para cada uno de los cambios realizados por el usuario. De esta forma es posible ampliar la funcionalidad de la herramienta proponiéndole, por ejemplo, al usuario la posibilidad de realizar un mismo cambio en distintas ubicaciones a la vez. El resultado del proceso de resolución de coincidencias es la creación de una nueva entrada (<RealContext>) con el contexto real del cambio resultante del proceso explicado anteriormente, y por tanto será este contexto el que DESK utilice para realizar posteriormente la modificación. La forma de encontrar el contexto apropiado se basa en el análisis y comparación de los distintos contextos encontrados para una sola primitiva, utilizando para ello el contexto sintáctico del modelo de monitorización.

A parte de esta heurística de desambiguación por matching del contexto, se utilizan otras técnicas que permiten, en casos muy puntuales, poder averiguar información sobre el origen del cambio realizado por el usuario, como por ejemplo:

- Desambiguación basada en estilos. Se aplica a casos en los que lo que el usuario pretende modificar son títulos, secciones o párrafos de una página. Esto se hace evaluando las etiquetas HTML que rodean al bloque sintáctico, y viendo si existen etiquetas que definan un encabezado, como por ejemplo <H1> o letras grandes con subrayados y negritas. Si esto ocurre, hay mucha probabilidad de que lo que se esté manipulando sea un título de una sección o página principal y por lo tanto es más apropiada la elección de un atributo `title` de un objeto como candidato a ser el <RealContext> para este caso de ambigüedad.
- Desambiguación basada en la topología de las relaciones en el modelo del dominio. Esta técnica consiste en evaluar la dependencia de un objeto con respecto a la red

semántica a la que pertenece, es decir, buscar su camino en la red y averiguar su proximidad al objeto principal extraído a través de la URL. En este caso se trata de aplicar un algoritmo de camino mínimo para ver a qué distancia están las distintas alternativas del objeto principal para evaluar un posible ganador (el objeto más cercano al principal) como candidato <RealContext> para deshacer la ambigüedad. Por ejemplo, en el caso de que un mismo literal se encuentre en dos atributos de dos objetos distintos, la expresión de caracterización "el título del prerrequisito del algoritmo de Dijkstra" primaría sobre "el nombre del tema relacionado del prerrequisito del algoritmo de Dijkstra", al encontrarse la primera más cerca del objeto principal, "El Algoritmo de Dijkstra", que la segunda.

Cuando a DESK le resulte imposible decidir cuál de los N contextos es el preferible para realizar la modificación, DESK recurre a la interacción con el usuario, presentándole en un formulario, en la página web de resultados, los posibles casos de ambigüedad para que sea el propio usuario el que se encargue de elegir cuál de las alternativas propuestas es la más adecuada.

Modelo de monitorización enriquecido

```
<MonitoringModel>
  <ApplyStyle ID="5" type="UNDERLINE">
    <TextAffected><![CDATA[Dijkstra's Algorithm]]></TextAffected>
    <InContext before="Relaxation Algorithm" after="Solo grafos dirigidos…"/>
      <FoundIn>
        <Class class_name="Algorithm" ID="Dijkstra" attribute="title" />
        <Class class_name="AtomicFragment"
              path="AtomicFragment,procedure,prerequisites,Algorithm,
                    items,Course,Domain"/>
      </FoundIn>
      <RealContext>

        <Class class_name="Algorithm" ID="Dijkstra" attribute="title" />
      </RealContext>
    <InContext>
  </ApplyStyle >

</MonitoringModel>
```

Figura 4.13: Caso resuelto de ambigüedad por múltiple contexto

4.5 Tratamiento de reglas de presentación PEGASUS bajo DESK

Como se ya se explicó en el capítulo anterior, las reglas de presentación de PEGASUS gobiernan aspectos dinámicos relacionados con la renderización o visualización de los distintos objetos del dominio, como la generación de enlaces, la correspondencia entre estilos de enlace y estados de objetos del dominio, la ordenación y disposición espacial de listas (de enlaces o fragmentos), y la generación de presentaciones predefinidas para subconjuntos de la red semántica como árboles y listas

encadenadas. Estas reglas consisten en cero o más condiciones seguidas por la presentación a aplicar cuando éstas se cumplen, descrita con la misma sintaxis de las plantillas. La aplicación de estas reglas es recursiva.

La gestión de reglas de presentación es una tarea delicada que requiere cierta familiaridad con el sistema. El cometido de DESK en este caso es el de gestionar los cambios, como hasta ahora se ha descrito, dentro del modelo de la presentación, teniendo en cuenta las reglas de presentación que también forman parte de dicho modelo. En la Figura 4.14 se muestra la relación entre DESK y los modelos subyacentes de PEGASUS, teniendo en cuenta la interacción del usuario a la hora de realizar cambios sobre una página web dinámica. La detección de reglas de presentación bajo DESK tienen la ventaja de que las reglas suelen tener un mayor nivel de abstracción que el resto de elementos de la presentación, y por tanto su reconocimiento y posterior tratamiento resulta mucho más práctico que modificar extensos fragmentos de presentación independientes que pueden aparecer dispersos por toda la presentación, es decir, con menor esfuerzo se pueden cambiar más cosas a la vez.

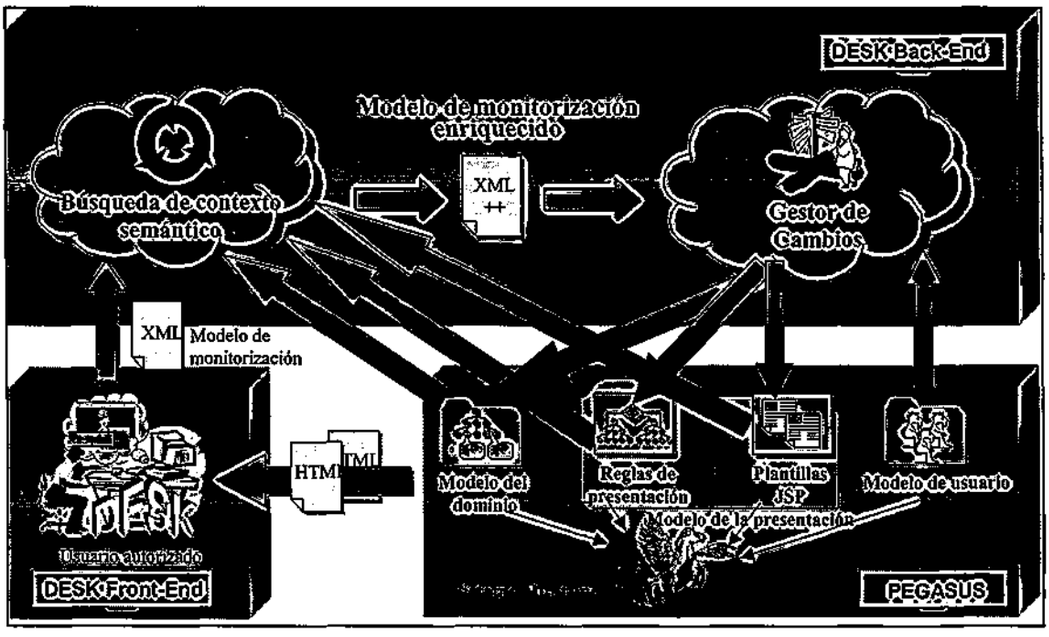


Figura 4.14: Relación entre DESK y los modelos de PEGASUS en el procesamiento de cambios en la presentación

El tratamiento de reglas de presentación es, sin embargo, una tarea difícil, si tenemos en cuenta que DESK sólo puede partir de lo que el usuario edita sobre el código HTML. Las reglas de presentación generan código dinámico de forma recursiva, y construir el camino inverso requiere de ciertas heurísticas para llegar a los modelos constructores a partir del resultado final generado. En concreto, DESK utiliza dos técnicas para hacer frente al tratamiento de las reglas de presentación. Una consiste en abstraer exactamente de dónde provienen los cambios a partir de la información existente en la presentación, es decir, las técnicas explicadas anteriormente para inferir

cambios en los modelos, y otra consiste en utilizar conocimiento adicional embebido proveniente de la generación dinámica de la página en PEGASUS.

4.5.1 Generación de semántica adicional en la presentación

En el capítulo anterior se habló sobre la gestión de reglas de presentación en PEGASUS, donde cada aparición de un objeto en la plantilla de la presentación se busca en el modelo del dominio y, a través de la aplicación de las reglas de presentación a ciertos objetos, se obtiene el documento hipermedia final.

Para que DESK pueda reconocer la procedencia de las reglas de presentación se necesita una adaptación de la arquitectura mostrada en el capítulo anterior (Figura 3.3) para que el propio motor de composición de páginas de PEGASUS pueda generar *meta-información* para identificar la procedencia del código que ha sido generado a partir de una regla. Esta modificación de la arquitectura se muestra en la Figura 4.15, donde se le ha añadido un módulo de anotación automática para la generación de meta-información al motor final de generación de documentos hipermedia. Este nuevo módulo es independiente de la funcionalidad explicada ya en PEGASUS, y podría integrarse como mecanismo *plug-in* a la arquitectura original descrita en el capítulo anterior.

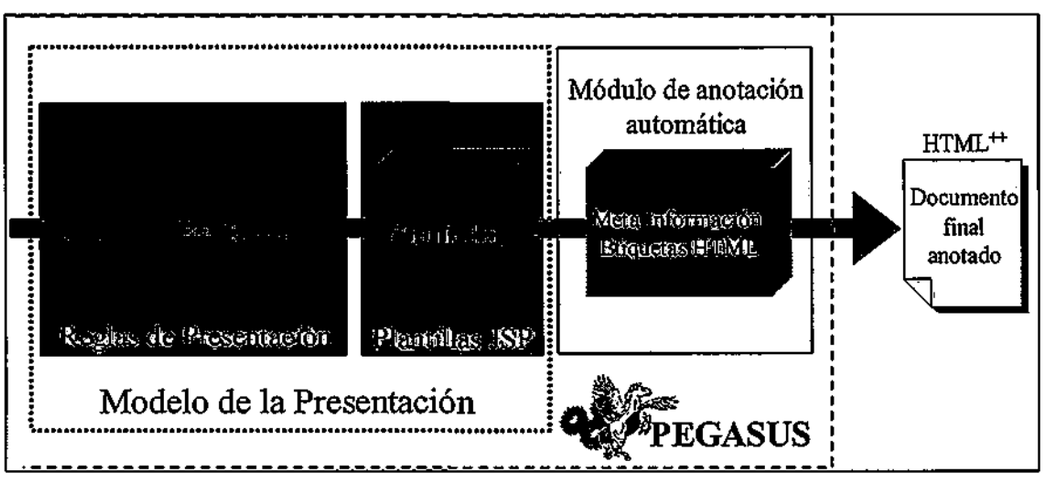


Figura 4.15: Ampliación del esquema de generación dinámica de documentos de PEGASUS (Figura 3.3) para la inserción de meta-información embebida en el documento HTML generado

4.5.2 Reconocimiento y tratamiento de meta-información bajo DESK

La meta-información generada por PEGASUS se compone básicamente de una serie de etiquetas adicionales, embebidas en el código HTML del documento final generado, presentes en ciertos controles HTML, como listas de selección, tablas, árboles, etc., y en general cualquier widget que sea susceptible de ser generado mediante reglas de presentación, y que codificarán información semántica relativa a la procedencia de dicho código, como el identificador de la regla que lo generó, el nombre de la clase, del objeto y de la relación, así como los atributos que intervienen. Es

conveniente decir que, aunque existen distintos tipos de reglas PEGASUS, DESK solamente tratará reglas que atañen directamente a la presentación, es decir, aquéllas que se encargan de generar todo tipo de widgets de presentación (tablas, listas, árboles, botones, etc.), reglas que afectan a la forma de presentar los datos (como el estilo de los links), el estilo de los widgets, etc. Esto resultará muy útil para la construcción de un asistente de edición, como se explicará más adelante, para detectar la intención del usuario al manipular widgets de presentación.

El utilizar meta-información, en vez de los mecanismos de inferencia explicados anteriormente, tiene su justificación en el empleo de reglas de presentación. La generación de meta-información es algo externo a PEGASUS y, por tanto, es un *plug-in* añadido para la localización de cierta información procedural con alto contenido semántico, esto hace que por otro lado la información generada sea menos transportable para realizar cambios sobre otros sistemas que no traten este tipo de meta-información.

En cualquier caso, DESK está preparado para trabajar con o sin meta-información, aunque en este último caso el tratamiento de reglas de presentación no es posible, lo que por contra aportaría una mayor compatibilidad con otros sistemas o modelos de presentación distintos a PEGASUS [MC03d].

En la Figura 4.16 se muestra un ejemplo de generación de meta-etiquetas para un control HTML (lista de selección), generado a partir de una regla (`R10`) de la plantilla de presentación. Como se ve en este ejemplo, la meta-información añadida en la página HTML generada consta del identificador de la regla (`ID="R10"`), el nombre del objeto principal (`main_object="Dijkstra"`) que se obtiene en la petición HTTP de la navegación, el nombre de la relación del dominio involucrada (`relation_name="prerequisites"`), así como el nombre de clase a la cual pertenece el objeto (`class_name="Algorithm"`). Igualmente, para cada uno de los elementos de la lista generada, se representa el nombre del objeto y el atributo que aparece como literal, de esta forma es muy fácil identificar exactamente de dónde provienen cada uno de los elementos generados por la regla de presentación, algo que sería muy difícil de obtener sin meta-información, debido a la estructura de grafo que tiene la red semántica asociada al modelo del dominio, así como al carácter recursivo de las reglas. Así pues, si el usuario decidiera desde DESK cambiar ciertas propiedades generales (como el color de un link, por ejemplo), DESK intentará identificar la procedencia de los datos así como la regla de presentación asociada, según el mecanismo explicado anteriormente, hasta dar finalmente con el modelo de la presentación donde se procederá a realizar los cambios oportunos.

Esta meta-información o meta-etiquetas no afectan para nada a la visualización del código HTML en los navegadores convencionales, al igual que tampoco supone ningún problema para el navegador de la aplicación cliente de DESK, el cual está preparado para la detección y tratamiento de este tipo de etiquetas.

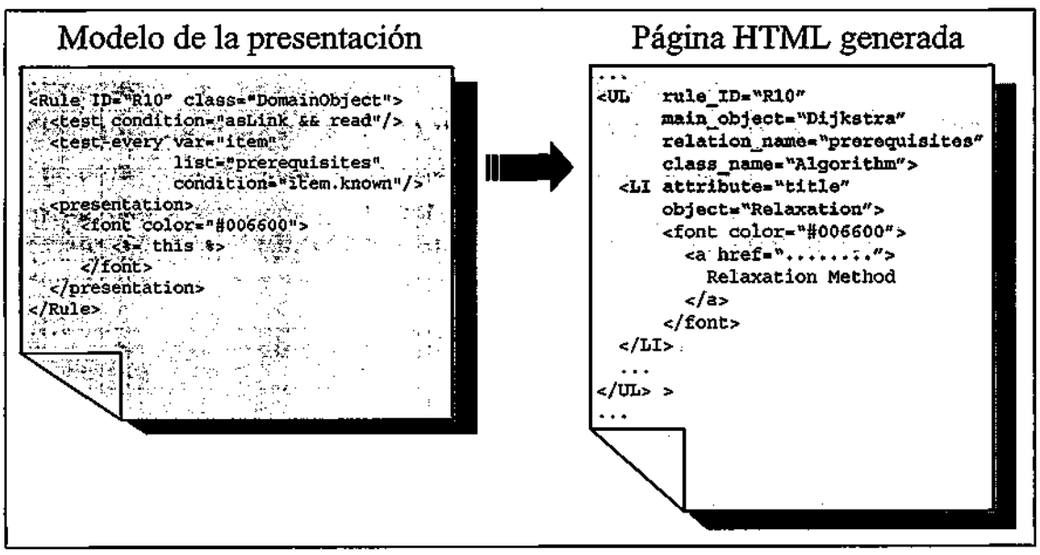


Figura 4.16: Generación de meta-información en reglas de presentación

Internamente, la meta-información procedente de PEGASUS es reconocida en DESK cliente mediante los algoritmos que componen las heurísticas de bajo nivel, en concreto dentro del módulo de identificación de estructuras especiales. Así pues, DESK comprueba la existencia de meta-información en las estructuras HTML que el usuario manipula, generando una entrada en el modelo de monitorización que contendrá la información semántica copiada directamente de las meta-etiquetas generadas por PEGASUS, en este caso el tratamiento en el servidor será mucho más simple, pues no hará falta buscar semántica asociada para la acción, sino proceder directamente a ejecutar el cambio realizado por el usuario con la información disponible en el modelo de monitorización.

4.6 Agente de inferencia DESK

La aplicación DESK cliente aporta un asistente personalizado capaz de inferir cambios de mayor alcance entre los objetos incluidos en la presentación. Estos cambios están relacionados con la manipulación de widgets, como la copia de elementos de un widget a otro, detectando la herramienta *patrones de iteración* que permiten completar estas secuencias, iniciadas por el usuario, de forma automática [MC03c]. Los patrones de iteración se definen como secuencias repetitivas de acciones o tareas de alto nivel que pueden darse dentro de un marco de trabajo, como por ejemplo la edición. Desde el punto de vista computacional es bastante útil detectar y modelizar estos patrones de iteración, de esta forma se pueden automatizar tareas de alto nivel que permitan guiar al usuario o ayudarle en los distintos procesos relacionados con la autoría. En este caso, y dado que DESK es un editor de documentos generados web dinámicamente, el marco de trabajo elegido será el de la edición, en concreto la edición de widgets HTML en documentos web.

La Figura 4.17 muestra una pantalla de DESK donde se da un ejemplo de patrón de iteración detectado al copiar elementos de una lista de elementos a una tabla. Para este tipo de casos lo que realmente se caracteriza es la copia reiterada de elementos, en este caso de una lista a una tabla, de forma que esta pueda ser automatizada por el sistema, completando él mismo directamente la secuencia en función de los parámetros iniciales: número de elementos de la lista y dimensión de la tabla creada por el usuario.

Como se vio en el Capítulo II, la extracción de conocimiento de páginas web es un campo relevante de trabajo hoy en día, en el que ha habido importantes aportaciones en lo relacionado con el reconocimiento y caracterización de tablas. A ese respecto, el agente de inferencia contribuye con una aportación notable mediante un mecanismo para reconocer y modelizar estructuras complejas de la interfaz que el usuario manipula, utilizando además información del dominio para caracterizarlas y ver su relación con la plantilla y la base de conocimiento. Esto permitirá, posteriormente, un tratamiento automático de las mismas por parte de las heurísticas de inferencia que DESK utiliza, llevando a término más fácilmente los cambios que el usuario realiza sobre la página web.

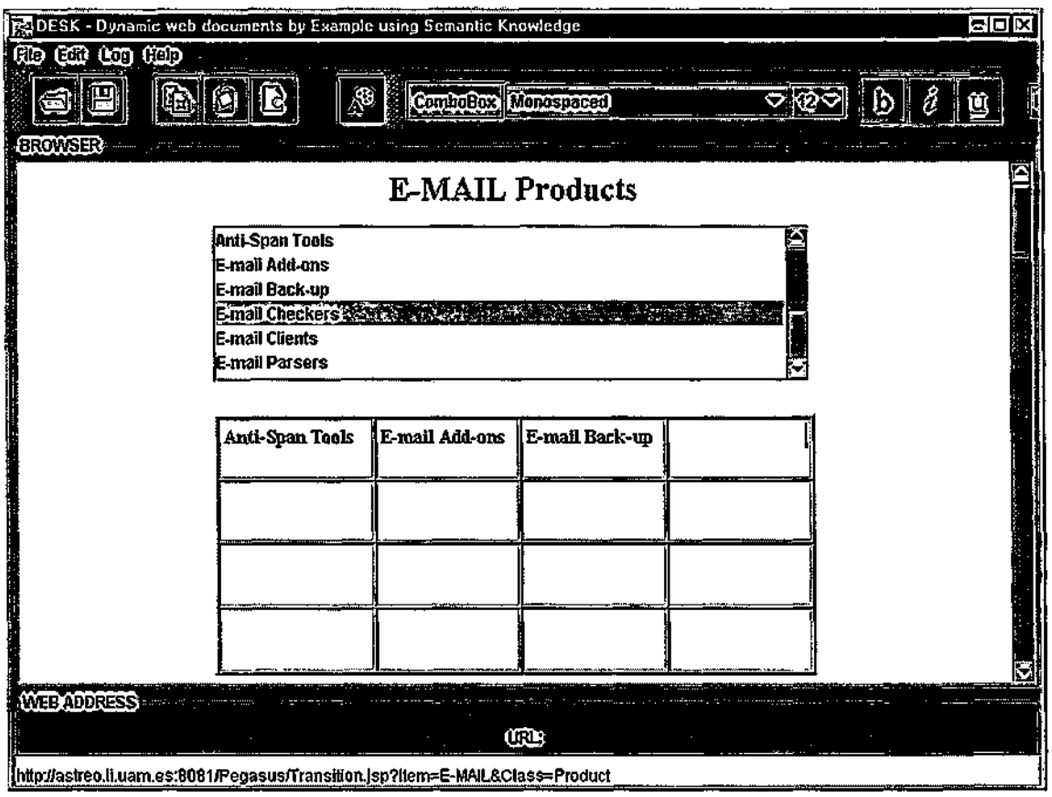


Figura 4.17: Copia de elementos de una lista de productos a una tabla que da lugar a un patrón iterativo

El asistente DESK se basa en la idea del Agente de Información [IAG], [BDP00], [BDP+00], utilizado en wrappers [GRV+98], [Mus99], [HS00], sin embargo en este caso el agente no busca dentro del código HTML, sino entre las acciones sobre

el propio código HTML realizadas por el usuario y reflejadas en un modelo más estructurado, como es el modelo de monitorización, del cual se puede obtener información inmediata sobre cualquier aspecto de la edición vinculada a las acciones llevadas a cabo por el usuario, codificadas como primitivas del modelo. De esta forma se podrán detectar posibles pistas de activación de heurísticas más complejas que permitan averiguar si es posible inferir un cambio sobre una interfaz HTML. Estos cambios incluyen, por ejemplo, transformar una tabla en una lista de selección, un combo box en una tabla, así como inferir otro tipo de acciones más complejas basadas en la detección de otro tipo de patrones, como el reordenamiento de listas, copiar atributos de una celda a otra en una tabla, etc. Los cambios se consideran atómicos, siendo responsabilidad del sistema detectar la relación entre los datos y los patrones iterativos a partir de la interacción parcial del usuario bajo la herramienta durante el proceso de edición. De esta forma, el agente comprende tres estados bien diferenciados:

- Pre-activación: se lleva a cabo a partir de la localización de ciertas primitivas clave en el modelo de monitorización (i.e. copia/pegado de elementos de un widget a otro), teniendo en cuenta además la propia configuración de activación del agente.
- Activación: donde el agente rastrea más concienzudamente el modelo de monitorización en busca de pistas más certeras sobre la información de pre-activación (i.e. creación de widgets involucrados), y activa las heurísticas apropiadas, en función de los parámetros analizados previamente, para la automatización final del proceso.
- Ejecución: donde se ejecutan las heurísticas que permiten la construcción de iteradores para automatizar tareas, actuando el agente como sustituto del usuario para completar el ciclo de iteración. Las heurísticas finalmente realizan las transformaciones correspondientes entre elementos de la presentación.

4.6.1 Configuración del agente

La principal misión del agente es la facilitar al usuario la edición del documento web mediante la automatización de ciertas tareas repetitivas o iterativas. Para ello, el agente busca en el modelo de monitorización primitivas que cumplan ciertos requisitos en función de una configuración establecida que puede ser definida por el usuario, la cual que se encarga de programar el comportamiento del propio agente. El fichero de configuración del agente codifica, a grandes rasgos, la siguiente información:

- Condiciones de pre-activación: Codificadas en cláusulas del tipo `<Condition>` que chequean la activación del agente ante la aparición de ciertas acciones presentes en el modelo de monitorización.
- Pool de patrones no regulares: Utilizado para codificar patrones de iteración definidos por el usuario (`<Pattern>`) y su función de resolución (`<Resolve>`) que permite la ejecución de heurísticas de transformación a partir de secuencias no

regulares, es decir, que no pueden ser obtenidas automáticamente mediante correspondencias lineales entre los atributos que definen la geometría del widget.

```
<TransformationHint searchLength="100">
  ...
  <widget type="List" changeTo="Table">
    <Condition action="Creation" widget="Table" />
    <Condition action="PasteFragment" from="Table" to="List"
     repeat="6" />
    <Non_Regular_Pattern_Pool>
      <Pattern   col_sequence="1,1,2,2" row_sequence="1,2,2,3"
                 elem_sequence="1,2,3,4">
        <Resolve i="from 1 to List.getSize()-1; i++1"
                 next_col_sequence="col[i], col[i]"
                 next_row_sequence="row[i], row[i+1]"
                 next_elm_sequence="elm[i]" />
      </Pattern>
      <Pattern   col_sequence="1,2,3,2,3,4"
                 row_sequence="1,1,1,2,2,2"
                 elm_sequence="1,2,3,4,5,6">
        <Resolve next_col_sequence="3,4,5,4,5,6,..."
                 next_row_sequence="3,3,3,4,4,4,..."
                 next_elm_sequence="7,8,9,10,11,..." />
      </Pattern>
      ...
    </Non_Regular_Pattern_Pool>
  </widget>
  ...
</TransformationHint>
```

El código anterior es un fragmento de la configuración del agente en formato XML. Los elementos `<TransformationHint>` especifican directrices que el agente tendrá que contemplar para pre-activar la detección de por ejemplo, en este caso, una transformación entre widgets (`<widget>`), de una lista de selección (`type="List"`) a una tabla (`changeTo="Table"`). Mediante la variable `searchLength` se le indica al agente el rango de inspección de, en este caso, 100 primitivas desde la última insertada en el modelo de monitorización. Cuando el agente detecta que en esas 100 primitivas se ha realizado una copia de elementos de una lista de selección a una tabla, la secuencia de comprobaciones de dicha transformación entre widgets empieza a ejecutarse. La primera comprobación será ver si se han creado anteriormente los widgets involucrados, para ello deben de satisfacerse las dos primeras condiciones `<condition>` de creación, una sobre la lista (`action="Creation" widget="List"`) y otra sobre la tabla (`action="Creation" widget="Table"`). Seguidamente se comprueba si existe una secuencia iterativa de copia de elementos de la lista a la tabla (`action="PasteFragment" from="Table" to="List"`) durante 6 ocasiones (`repeat="6"`). Si las condiciones anteriores se cumplen, se procederá a mostrar un mensaje al usuario para confirmar si se desea cambiar, de forma automática, la lista por la tabla. Posteriormente a esta confirmación, el usuario también puede elegir si prefiere conservar la geometría de la tabla definida inicialmente, o si permite que el sistema pueda crear nuevas filas y/o columnas en caso de ser necesario. En la Figura

4.18 se muestra un ejemplo de este proceso, donde se ve cómo DESK ha detectado el patrón de iteración durante las 6 acciones de copia de elementos de la lista a la tabla. Una vez analizados los widgets involucrados, el agente procede a preguntar al usuario si desea la transformación automática entre widgets, ejecutando las heurísticas correspondientes para concluir, finalmente, sustituyendo la lista de selección por una tabla que contiene exactamente los mismos elementos.

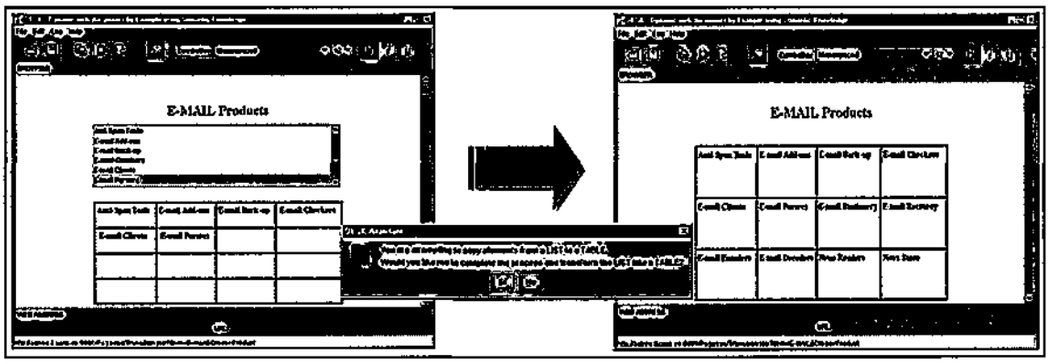


Figura 4.18: Detección bajo DESK de un patrón de iteración a partir de acciones de edición, y transformación final de una lista en una tabla

4.6.2 Detección de patrones iterativos

Una vez que el usuario ha contestado afirmativamente a la propuesta de la herramienta de realizar una transformación automática entre widgets, el agente entra en fase de ejecución invocando heurísticas que determinan el mecanismo de detección patrones de iteración, que tiene como finalidad el completar la secuencia de inserción de elementos desde el widget origen hasta el destino, respetando la propuesta inicial del usuario con respecto a la disposición de los elementos. En este caso, un patrón de iteración se define como una secuencia de acciones de edición relacionadas sobre una parte integrante o estructura del documento web, realizada por el usuario de forma iterativa y pudiendo ser detectada para automatizar el proceso por parte de la herramienta. Para ello se necesita establecer una relación entre la tarea de alto nivel que se repite y la parte o bloque del documento sobre el que actúa, es decir, se utiliza el modelo de monitorización rico en información de este calibre.

La detección de patrones iterativos es, sin embargo, una tarea compleja que otros sistemas resuelven utilizando algoritmos de aprendizaje o modelos probabilísticos. En este caso se emplearán dos técnicas, una para patrones regulares, mediante algoritmos especializados de detección de secuencias lineales y otro para el tratamiento de patrones no regulares, donde se permite al usuario el definir secuencias más complejas a partir de un *pool* de patrones con el que se realizará *matching*.

Desde una perspectiva general, el algoritmo para la selección de un tratamiento regular o no regular de los patrones de iteración es muy simple. Si el patrón con el que se hace matching se encuentra en el pool, se procederá a utilizar la información de dicho

patrón para completar la secuencia, si no, se aplicarán los algoritmos para patrones regulares que intentarán crear un modelo de inducción para completar la secuencia iniciada por el usuario.

4.6.2.1 Patrones regulares

Las técnicas para el tratamiento de patrones de iteración regulares comprenden una serie de heurísticas que serán ejecutadas automáticamente por el agente después satisfacer las condiciones de pre-activación y adecuación según la configuración del mismo. En estos algoritmos, denominados algoritmos IP[1], se identifica automáticamente de qué tipo son los widgets sobre los que se está actuando, obteniendo unos valores u otros en consecuencia y permitiendo un alto grado de generalidad en el tratamiento de patrones iterativos. Aunque cada algoritmo IP está especializado en el tratamiento de un widget en concreto, en esta tesis se tratará el caso de estos algoritmos para el tratamiento de tablas y estructuras lineales (como listas y cobo boxes), siendo muy fácil la creación de otros algoritmos especializados para el tratamiento de otras estructuras menos regulares (p.e. árboles).

El cometido que persiguen el conjunto de heurísticas que forman los algoritmos IP, es el de construir un modelo lineal de iteración para poder completar la secuencial iniciada por el usuario y que dio lugar a la detección de un patrón iterativo por parte del agente. A continuación se muestra un pseudocódigo general perteneciente a un fragmento de estos algoritmos para el caso de la transformación entre widgets, de lista de selección a tabla:

```
ColumnSequence     = A.getColumnSequence(W2);
RowSequence        = A.getRowSequence (W2);
ElemIndexSequence = A.getElementIndexSequence (W1);

ColJumpSet         = ColSequence.getColJumpSet(0);
RowJumpSet         = RowSequence.getRowJumpSet(0);

ColShiftSet        =
      BuildColShiftSet(ColumnSequence,ColJumpSet,RowJumpSet);
RowShiftSet        =
      BuildRowShiftSet(RowSequence,ColJumpSet,RowJumpSet);

Iterator           = BuildIterator(W2.getBounds(0),TG,ColShiftSet,
                     RowShiftSet,ElemIndexSequence);
...
While (Iterator.hasNext(0)) {
   i = Iterator.getNexti(i);
   j = Iterator.getNextj(j);
   k = Iterator.getNextk(k);
   W2.setElementAt(i,j,W1.getElementAt(k));
}
```

En el código anterior `W1` representa el widget origen, en este caso una lista, `W2` el widget destino, en este caso una tabla y `TG` representa el conjunto de elección del

[1] Iteration Patterns, o patrones de iteración. Así se definen en DESK el conjunto de algoritmos utilizado para reconocer patrones de iteración sobre widgets predefinidos de la presentación.

usuario para respetar la geometría inicial del widget destino o ampliarlo en cuanto, en este caso, a filas y columnas. A es el conjunto de primitivas capturadas y que representan la copia de elementos (6 veces, según el ejemplo visto anteriormente) que se ha producido entre el widget W1 y el W2. La información de este conjunto es bastante relevante, puesto que de él se extraerán características comunes que permitan crear modelos abstractos de ambos widgets. De esta forma será posible el acceder a características comunes sobre dicho conjunto mediante los operadores:

```
A.getWidgetType(Widget)                A.getID(Widget)
A.getSize(Widget)                      A.getClassName(Widget)
A.getElementIndexSequence (Widget)     A.getObjectName(Widget)
A.getColumnSequence(Widget)            A.getRelationName(Widget)
A.getRowSequence (Widget)              A.getExistsRelation(Widget1,
A.getElementAt(Widget,i[,j])                              Widget2)
```

La principal utilidad de estos operadores es modelizar los widgets en función de su geometría, el tipo de elementos que contienen, etc., y obtener así algunos datos sobre la dinámica global de las acciones sobre los propios widgets, como la secuencia de filas y columnas de inserción, los saltos entre filas y columnas, etc. De esta forma será posible la construcción de un iterador (Iterator) que permita ir obteniendo los índices de inserción y extracción de un widget a otro mediante la construcción de una máscara cíclica de incrementos obtenidos a partir de cálculos sobre los datos extraídos del conjunto A.

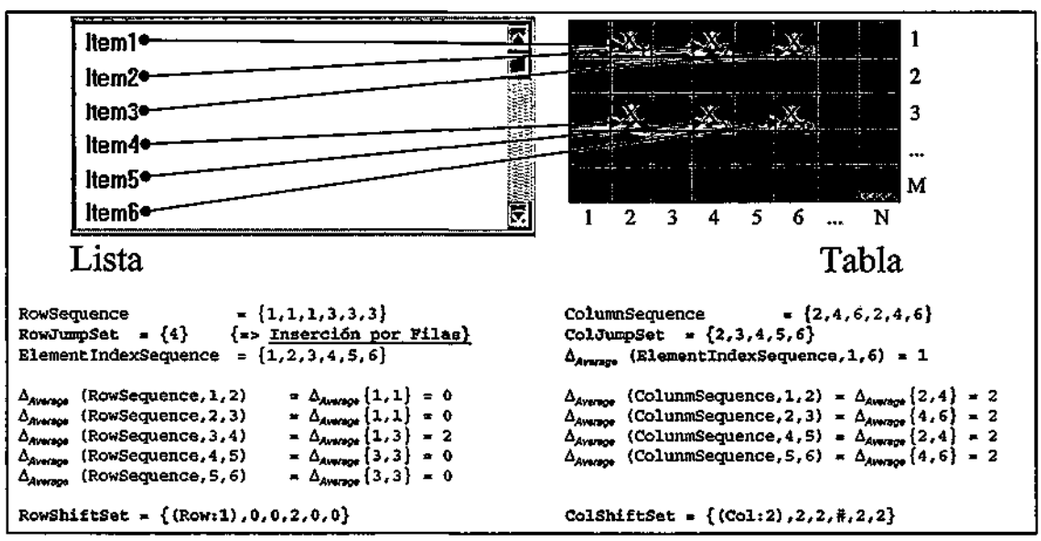


Figura 4.19: Ejemplo de actuación del algoritmo IP para tablas

En la Figura 4.19 se muestra un ejemplo de lo explicado, teniendo en cuenta además el código anterior. Aquí se refleja cómo el usuario ha copiado 6 elementos de una lista a una tabla, y cuál es el contenido de los conjuntos utilizados en el algoritmo que ejecuta el agente. Los conjuntos ColumnSequence y RowSequence reflejan la secuencia de inserción llevada a cabo por el usuario en la tabla, el algoritmo además calcula los conjuntos de salto entre filas (RowJumpSet) y entre columnas

(`ColJumpSet`). Nótese que el algoritmo detecta si la inserción se ha realizado por filas o por columnas. Para ello solamente tiene que comparar los conjuntos de salto, de forma que si `RowJumpset` es mayor en tamaño que `ColJumpSet`, la inserción se ha realizado por columnas, en caso contrario la inserción se realizaría por filas, y sin son iguales habría que tomar consideraciones especiales, pues esto indicaría que existe una relación lineal entre filas y columnas. Seguidamente se calcula una máscara de incremento de columnas (`ColShiftSet`) y de filas (`RowShiftSet`), a partir de un operador denominado $\Delta_{Average}$, cuya fórmula se detalla a continuación:

$$\Delta_{Average}(x_1,x_2,x_3,...,x_n)=\begin{cases}\dfrac{(x_2-x_1)+(x_3-x_2)+...+(x_n-x_{n-1})}{n-1}, & n>1\\ 0, & n\leq 1\end{cases}=\begin{cases}\dfrac{x_n-x_1}{n-1}, & n>1\\ 0, & n\leq 1\end{cases}$$

Este operador halla los incrementos parciales medios para los conjuntos de secuencia de cualquier widget, en el caso del ejemplo anterior entre columnas, filas y la secuencia de elementos de la lista, y se aplica teniendo en cuenta las rupturas de secuencia reflejadas en los conjuntos de salto. El objetivo que alcanza este operador es obtener un par de máscaras (conjuntos `ColShiftSet` y `RowShiftSet`) que detallan los incrementos a seguir de forma cíclica, utilizando saltos de fila y de columna para la ruptura de secuencia, y teniendo en cuenta la posición inicial de la inserción (`Col:2` y `Row:1`) a partir de la cual se irán realizando los incrementos de los índices: de 2 en 2 columnas para la primera fila, saltando luego dos filas más (# en `RowShiftSet` y 2 en `ColShiftSet`) y luego de nuevo de dos en dos columnas. Todos los conjuntos calculados anteriormente se utilizan para construir finalmente un iterador, encargado de proporcionar mecanismo utilizado para componer un bucle sobre los índices de recorrido de los widgets. El iterador se encarga de controlar los límites de la geometría de la tabla y de crear nuevas celdas en caso de que el usuario así lo haya elegido. También controla la secuencia de extracción de elementos del widget origen, en este caso a la secuencia de elementos de la lista se le aplica el operador $\Delta_{Average}$, dando como resultado que los elementos se copiarán en orden de 1 en 1. De esta forma la herramienta puede construir un proceso iterativo para completar la secuencia inicial proporcionada por el usuario al copiar y pegar los datos de un widget a otro.

En la Figura 4.20 se muestran varios ejemplos de tratamiento de distintos tipos de patrones que el agente puede reconocer, inclusive aquellos que contienen rupturas asimétricas de columnas y filas (*cut-in*), y donde además el sistema puede averiguar la dinámica de la inserción, por filas o por columnas, creando distintas máscaras de inducción en cada caso (& para saltos de columna y # para saltos de filas). También se detectan casos en los que la dinámica es una función igualdad, existiendo el mismo número de saltos de filas que de columnas.

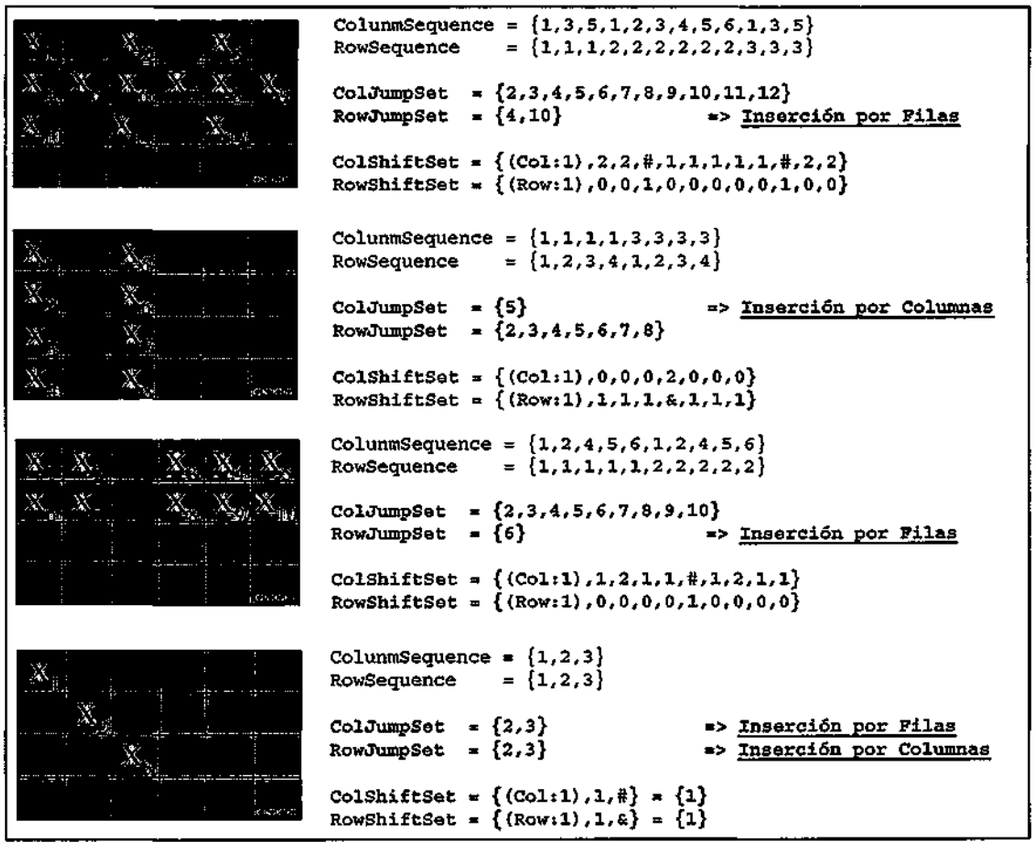


Figura 4.20: Distintos ejemplos de patrones de iteración detectados por el agente de inferencia DESK

4.6.2.2 Patrones no regulares

Sin embargo, no siempre se puede crear un modelo de inducción para cualquier tipo de patrón. Si el patrón de iteración en sí no es demasiado regular (ver Figura 4.21), es imposible construir un iterador para recorrer la estructura y recomponer la secuencia inicial ideada por el usuario. En cualquier caso, DESK hace frente a este problema permitiendo la definición de un pool de patrones que puede ser definido por el usuario y que permitirá codificar cualquier tipo de patrón, inclusive combinaciones ad-hoc definidas por el propio usuario. En el código XML de configuración del agente visto anteriormente, la etiqueta `<Non_Regular_Pattern_Pool>` detona el pool de patrones no regulares, donde en su interior se pueden definir patrones (`<Pattern>`) a detectar junto con la secuencia utilizada para completar la iteración (`<Resolve>`). Para definir patrones de esta forma se pueden utilizar dos notaciones distintas, una notación abstracta a partir de un índice de recorrido, o una notación más natural indicando exactamente la secuencia completa a seguir por el agente. En la Figura 4.21 se presentan ejemplos de patrones de este tipo, donde los dos primeros (a y b) se corresponden con los definidos en el código de configuración del agente en forma de (a) una notación abstracta, representada por la iteración sobre los elementos de la lista (`<Resolve i="from 1 to List.getSize(); i++1"`) y (b) una notación

donde se especifica la secuencia completa a seguir por el agente (`Resolve next_col_sequence="3,4,5,4,5,6,..."`).

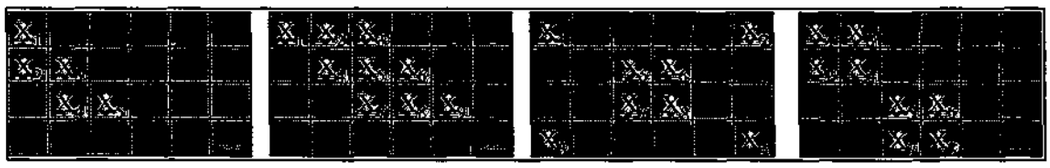


Figura 4.21: Ejemplos de patrones de iteración no regulares; a, b, c y d, de izquierda a derecha

El agente puede interpretar indistintamente patrones codificados en ambas notaciones, siendo la segunda de ellas la más idónea para patrones muy específicos y que no obedecen a ningún tipo de relación establecida entre filas y columnas. El agente se encarga de realizar *matching* con cada uno de los patrones, eligiendo uno u otro en función de la similitud con respecto a la secuencia inicial del usuario. En este caso el valor de repetición (`repeat="N"`) juega un papel importante, pues cuanto más grande sea el valor de la secuencia a observar por el agente, mayor será la precisión al elegir uno u otro patrón del pool.

4.6.3 Ejecución de los cambios detectados por el agente DESK

El resultado final del proceso de detección de patrones iterativos, si éste se lleva a cabo con éxito, es la sustitución de un widget de tipo lista por otro de tipo tabla. A bajo nivel en DESK, este tipo de cambio se codifica de forma similar a otros cambios que pudiera realizar el usuario, aunque no los realice directamente él. La forma de codificar el cambio es mediante la correspondiente primitiva en el modelo de monitorización, como la que se muestra a continuación:

```
<ChangeWidget ID="3">
  <From type="list"    id="L01"    relation="subCategories"
        class="HigherCategory"     objectID="Internet"
        ElemShiftSet="1"/>
  <To   type="Table"   id="T01"    relation="subCategories"
        class ="LowerCategory"     objectID="Internet"
        ColShiftSet="(Col:1),1,#,1" RowShiftSet="(Row:1),0,1,0" />
</ChangeWidget>
```

En este código puede verse una nueva primitiva (`ChangeWidget`) la cual refleja la semántica del cambio directamente en DESK cliente, especificando los valores del dominio asociados a ambos widgets, y añadiendo incluso información sobre la máscara de iteración para poder reproducir el cambio de la misma forma en la parte del servidor, después de haber procedido DESK cliente a llevar a cabo visualmente el cambio en la herramienta de edición del cliente. La primitiva anterior supone que los dos widgets han sido creados anteriormente por el usuario, del resto de pasos ya se encarga directamente el sistema mediante los ya mencionados mecanismos de inferencia. Esta modelización automática del proceso permite tratar distintos tipos de widget de forma general y escalable, si bien es verdad que algunas de estos widgets

tienen mayor complejidad en su tratamiento, como árboles y listas, respecto a otras que son más triviales de tratar mediante este mecanismo, como las listas de selección y los combo boxes.

Tal y como se explicó anteriormente, la inclusión de cierta meta-información o meta-etiquetas en este tipo de estructuras de la interfaz facilita llevar a cabo cambios como las transformaciones descritas anteriormente, los cuales poseen un mayor nivel de abstracción. En el caso anterior, como lo que se trataba era de sustituir un widget por el otro, la herramienta mantiene intacta la procedencia de los datos, es decir, las referencias al modelo del dominio y a la ontología, así como las correspondientes vinculaciones en el modelo de la presentación para poder efectuar los cambios. En la Figura 4.22 se muestra, a partir de la primitiva modificada anterior, cómo el sistema localiza el widget que representa la lista de selección, y procede finalmente a cambiarlo por una tabla, una vez enviado el modelo de monitorización a la aplicación DESK servidor y procesado éste posteriormente. El resultado final de este proceso bajo DESK queda reflejado en la Figura 4.18 vista con anterioridad.

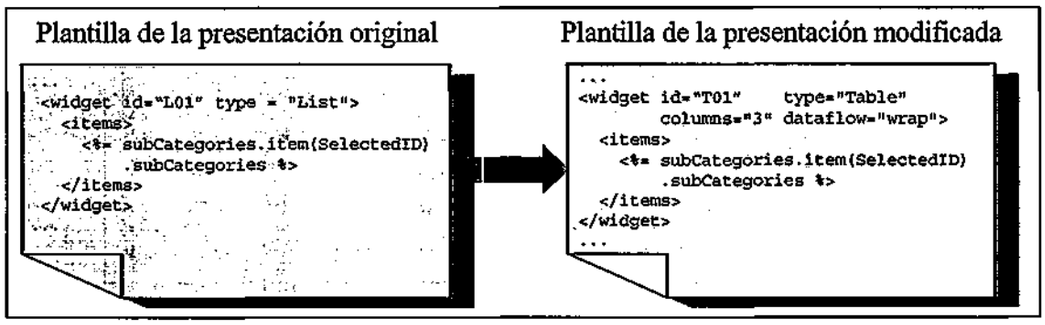


Figura 4.22: Cambio final llevado a cabo por DESK servidor al final del proceso de detección de patrones iterativos

4.7 Extensiones no WYSIWYG

Hasta ahora se han venido explicando los distintos aspectos de DESK en función de la capacidad de PEGASUS para las páginas web dinámicas que el usuario edita. Esto implica, groso modo, que debe de existir una correspondencia entre lo que PEGASUS genera y lo que DESK manipula.

Una de las características presentes en PEGASUS es la de poder tratar dependencias entre el conocimiento del dominio y los modelos del usuario y de la plataforma, como ya se vio en el capítulo anterior. DESK puede llevar a cabo la autoría de estas dependencias, permitiendo al usuario crearlas a medida a partir de los datos implícitos en la presentación pertenecientes al modelo del dominio. Esto conlleva una ampliación respecto a las posibilidades planteadas inicialmente sobre DESK, permitiendo así realizar una autoría de la adaptatividad de las páginas generadas dinámicamente, siendo la propia herramienta la encargada de inferir directamente los cambios sin requerir de mucho esfuerzo por parte del usuario final.

A pesar de que esta discusión será expuesta en el próximo capítulo, en concepto de trabajo futuro o ampliación de la herramienta, es conveniente decir que DESK permite, utilizando el formalismo propuesto, llevar a cabo la adaptatividad en función de modelos abstractos que, en un futuro, deberían estar disponibles en tiempo de diseño (modelos del usuario y de la plataforma). En cualquier caso, la idea principal que motivó este trabajo es la de alejarse lo menos posible del principio WYSIWYG, evitando en todo momento la utilización de modelos con mayor nivel de abstracción que podrían dificultar su entendimiento por parte de usuarios poco adiestrados. Esta premisa limita, por otro lado, la capacidad expresiva de DESK, impidiéndole manipular directamente las correspondencias visuales entre objetos del dominio, es decir, que el usuario pueda trabajar visualmente con la plantilla de la presentación y las referencias en la base de conocimiento. En cualquier caso, esta decisión permitirá estudiar cómo de lejos se puede llegar sin abandonar el WYSIWYG, siendo obvias las limitaciones inherentes que impiden el escalar DESK con diferentes vistas de modelos abstractos para aumentar su expresividad, en compromiso con la facilidad de uso actual de la herramienta [MC03b]. Esto es lo que se domina el *gentle slope* de la complejidad en la creación de interfaces de usuario [KF03] buscando, en el caso de DESK, un equilibrio aceptable entre expresividad y facilidad de uso.

A parte de las acciones estrictamente WYSIWYG de edición en DESK, existen otras que permiten a un diseñador más experimentado, y con autorización para ello, definir y modificar condiciones sobre la visualización de ciertos fragmentos de la presentación a partir de un perfil o modelo del usuario, es decir, permitir adaptar la presentación a perfiles muy concretos en un momento dado, dependiendo de quién sea el usuario que está interactuando con el sistema. Para ello, DESK cuenta con la ayuda de HADES, que es realmente el gestor de los perfiles de usuario durante la interacción. Esta misma información, como se vio anteriormente, servirá también para ser tenida en consideración por PEGASUS a la hora de renderizar los objetos del dominio en la presentación a partir de las reglas de presentación.

Las primitivas que representan condiciones sobre el perfil del usuario en un fragmento HTML, poseen un mayor nivel de abstracción que las que se han visto anteriormente, y por el momento no pueden considerarse WYSIWYG, pues habría que tener en cuenta el valor actual del perfil o el modelo del usuario en tiempo de diseño. Esto podría contemplarse no obstante como mejora de la herramienta, permitiendo realizar distintos *previews* a partir de una simulación mediante un perfil concreto del usuario en cuestión, tal y como se hace en herramientas como en [MC00]. La forma de poder crear condiciones sobre un texto HTML es, sin embargo, una acción muy simple que es tratada bajo DESK utilizando los mecanismos de inferencia ya conocidos hasta ahora. En la Figura 4.23 se muestra una pantalla de DESK, a partir de la cual se pueden establecer distintas condiciones sobre el modelo del usuario y de la plataforma, gestionados internamente por HADES. En esta ventana se especifican las condiciones sobre los atributos del modelo (p.e. *Student.Programming Skills.OOP = yes*) en forma

de predicado lógico con estructura de conjunción de disyunciones o forma normal conjuntiva. De esta forma se pueden añadir tantas cláusulas AND y OR como se quiera para cada una de las condiciones establecidas. Un diseñador solamente tendría que seleccionar el texto que quiere condicionar y después introducir el predicado lógico correspondiente en la ventana anterior. Una vez hecho esto DESK crea una primitiva en el modelo de monitorización con su contexto sintáctico y cierta semántica asociada sobre la condición establecida bajo el HTML seleccionado, enviando esta información al servidor para que el cambio surta efecto. DESK detecta si el texto ya estaba sobre alguna condición anterior, de esta forma el diseñador podría modificar dicha condición cuantas veces quiera.

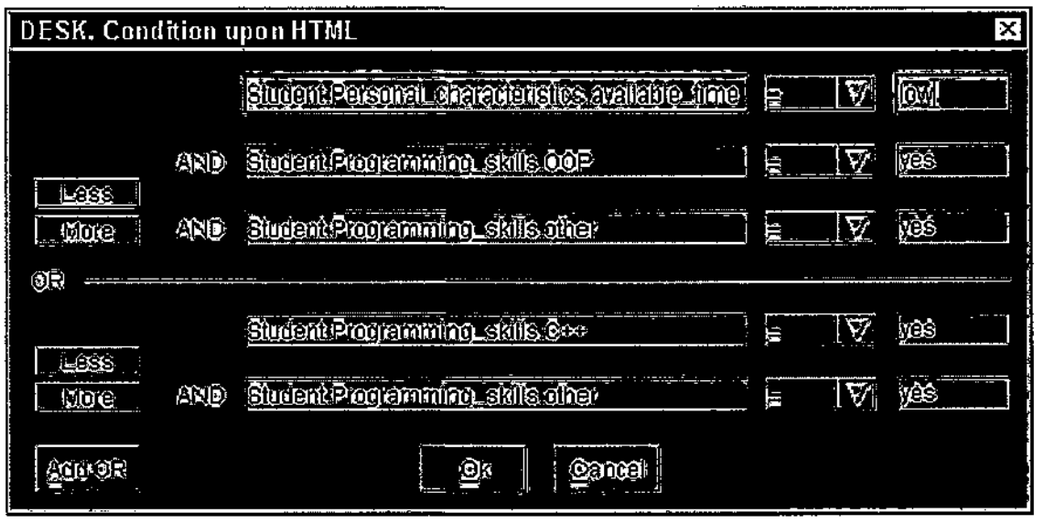


Figura 4.23: Creación de una condición sobre la visualización de un fragmento HTML

En la Figura 4.24 se muestra un caso de ejemplo donde (1) el diseñador bajo DESK selecciona un texto y crea una condición para que dicho texto se visualice sólamente si el usuario es mayor de edad. De esta forma (2) DESK crea una entrada en el modelo de monitorización reflejando esta acción. En este caso la condición de visualización del texto es incluida en la primitiva, junto con la información restante que se compone del texto seleccionado y del contexto sintáctico para su posterior localización. Una vez que el modelo de monitorización ha sido enviado al servidor (3), la aplicación DESK servidor localiza en el modelo del dominio el correspondiente contexto semántico para luego (4) poder realizar el cambio en la plantilla de la presentación.

Estos cambios en las plantilla resultan fáciles de realizar y de interpretar posteriormente debido a que, bajo PEGASUS, las plantillas del modelo de la presentación se codifican utilizando el lenguaje JSP, y por tanto es posible aprovechar la potencia del lenguaje JAVA para crear este tipo de condiciones que se auto-evaluarán sin necesidad de ninguna interpretación o tratamiento previo.

Por otro lado, en el caso de que el diseñador quiera modificar una condición ya existente sobre un fragmento HTML, solamente tendrá que hacer doble click y editar, en la venta de la Figura 4.23, las condiciones que desee. DESK reconoce, mediante el módulo de tratamiento de estructuras especiales que se encuentra dentro de las heurísticas de bajo nivel, las condiciones ya definidas con anterioridad. Para ello DESK vuelve a hacer uso de la meta-información transmitida por PEGASUS, que le informará si un fragmento ya tenía una condición declarada para que DESK pueda reconocer este hecho, guardando el identificador y los objetos del dominio relacionados para así contextualizar posteriormente el cambio en el servidor. De esta forma podrá detectarse la condición existente con el fin de modificarla con los cambios realizados por el diseñador bajo la herramienta.

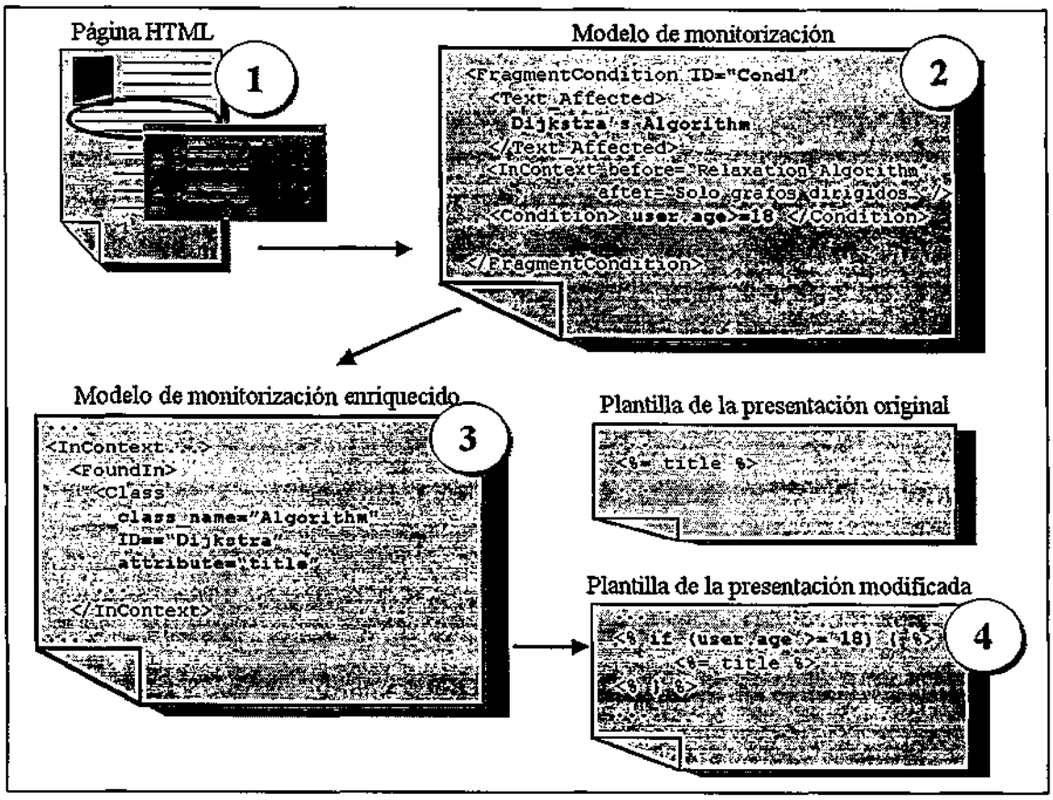


Figura 4.24: Ejemplo de creación de una condición sobre un fragmento HTML utilizando DESK

4.8 Un ejemplo completo

A continuación se ilustrarán las técnicas descritas en este capítulo y el anterior (Capítulo III. Generación de documentos web dinámicos) mediante un escenario de uso. Se trata de proporcionar un "escaparate" sencillo para una tienda *on-line* de productos software y equipos informáticos, como en TUCOWS.

Para el recorrido por las etapas en la elaboración del ejemplo se utilizarán las herramientas explicadas hasta ahora, distinguiendo entre dos fases bien diferenciadas:

1) La construcción del modelo del dominio, utilizado en PEGASUS, por un diseñador mediante la herramienta PERSEUS.
2) La modificación de la interfaz generada a partir de los cambios realizados por un usuario utilizando la herramienta DESK.

La herramienta HADES estará presente en todo momento para la integración de los distintos pasos y herramientas descritos.

4.8.1 El ejemplo propuesto

Se utilizarán los mínimos requisitos necesarios para no complicar en la medida de lo posible el desarrollo del ejemplo y evitar así distraer al lector con detalles irrelevantes, siendo el cometido principal mostrar de una forma práctica los mecanismos utilizados en las distintas herramientas descritas.

La aplicación web propuesta está basada en una tienda electrónica que ofrece diferentes productos, agrupados en niveles de categorías de forma que un usuario pueda navegar por estas categorías, permitiéndole visualizar características más detalladas de cada uno de los ítems que se muestran en cada categoría.

La tienda tiene varios catálogos de productos (p.e. Software, Portátiles, Periféricos, etc.), determinadas categorías principales de productos (p.e. Internet, Multimedia, Editores, etc.), distintas sub-divisiones de categorías (p.e. Navegadores, Correo Electrónico, etc.), diferentes sub-categorías (p.e. Clientes de Correo Electrónico, Parsers de Correo Electrónico, etc.) y, finalmente, una variedad productos comerciales relacionados con las categorías mencionadas (p.e. para el caso de Clientes de Correo Electrónico, productos como Allegro, Eudora, etc.). De esta forma, se pretende una navegación jerárquica típica en este tipo de tiendas electrónicas, donde el usuario se mueve, partiendo de un catálogo, por las distintas categorías y sub-categorías hasta llegar al producto que realmente le interesa.

4.8.2 Creación de la ontología y los objetos del dominio

El ejemplo descrito puede ser modelizado en PEGASUS mediante una ontología o taxonomía. De esta forma, la codificación de los distintos productos y categorías se llevará a cabo mediante una descomposición jerárquica, a partir de unas dependencias establecidas entre cada una de las entidades. Esta estructura se muestra en la Figura 4.25, donde se refleja la descomposición jerárquica narrada anteriormente a partir de los distintos catálogos, categorías y productos que forman el ejemplo de uso.

Una vez que se ha pensado en una estructura de la ontología, el siguiente paso será llevar a cabo dicho diseño utilizando la herramienta PERSEUS (Figura 4.26). Así pues, el autor comenzaría por diseñar las clases que van a formar la ontología del dominio, para acabar definiendo las propias instancias de las clases y las relaciones entre ellas, dependiendo del número de productos y categorías que se vayan a necesitar.

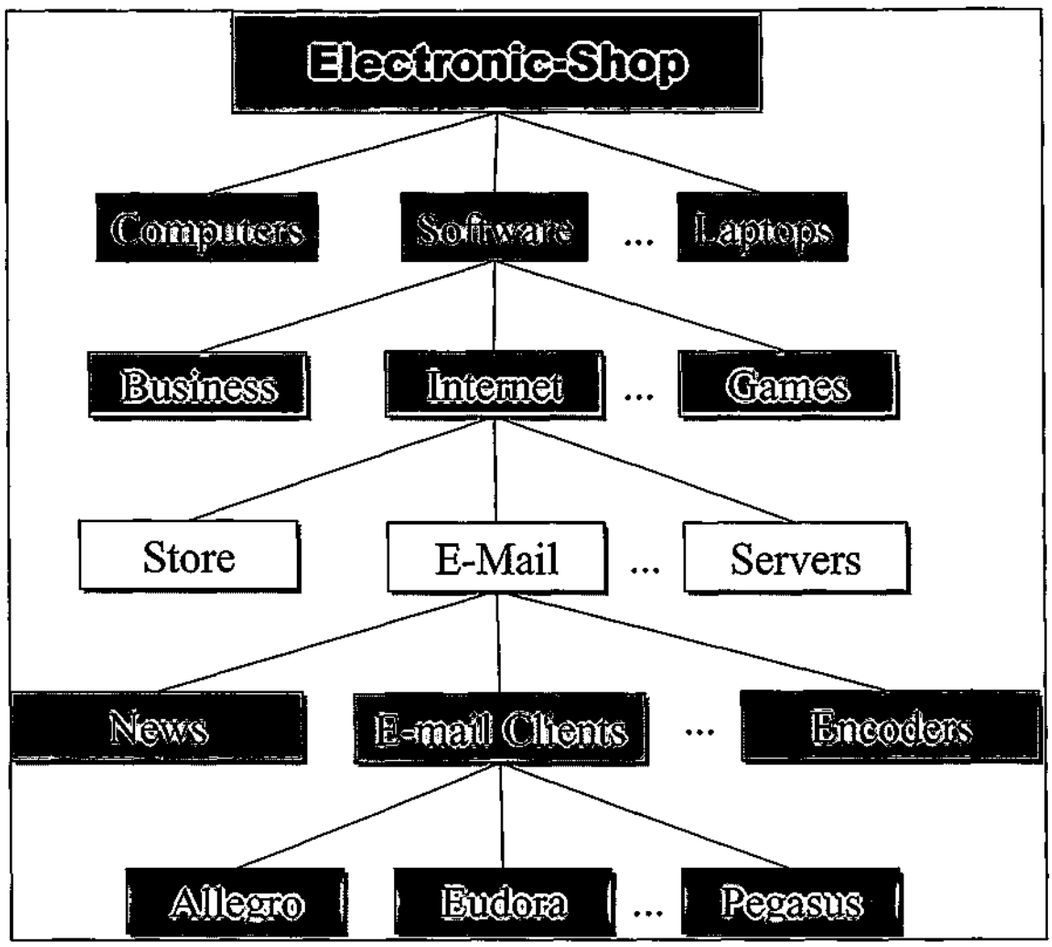


Figura 4.25: Jerarquía de componentes para la creación de una tienda electrónica basada en la web

Las clases definidas para la composición de la ontología del dominio pueden verse en la Figura 4.27, donde se han creado cuatro clases principales, denominadas `Catalog`, `HigherCategory` para categorías principales, `LowerCategory` para lo que antes hemos llamado sub-categorías, y finalmente `Product` para contemplar los distintos productos que formarán la tienda. Además se han definido las correspondientes relaciones entre cada una de las clases de la ontología, permitiendo así la construcción del grafo que formará la red de objetos del dominio. Algunas de estas relaciones corresponden a dependencias del tipo `BelongsTo`, en el caso de Producto, para representar la bidireccionalidad en el grafo de dependencias entre objetos del dominio. Otras relaciones como `Information` permitirán almacenar contenidos extras para cada uno de los productos. También, para cada una de las clases, se definen atributos específicos como, en el caso de `Product`, el *título*, el *precio*, la URL de acceso, etc. A parte de las clases mencionadas, serán necesarias otro tipo de clases que se utilizarán como ayuda al diseñador en la construcción de objetos, como es el caso de `Fragment` y `AtomicFragment`, clases padre e hijo respectivamente. Estas dos clases permitirán

definir fragmentos de código HTML para la inclusión directa de código con etiquetas e incluso URL con direcciones a recursos multimedia, como fotos, sonidos, video, etc.

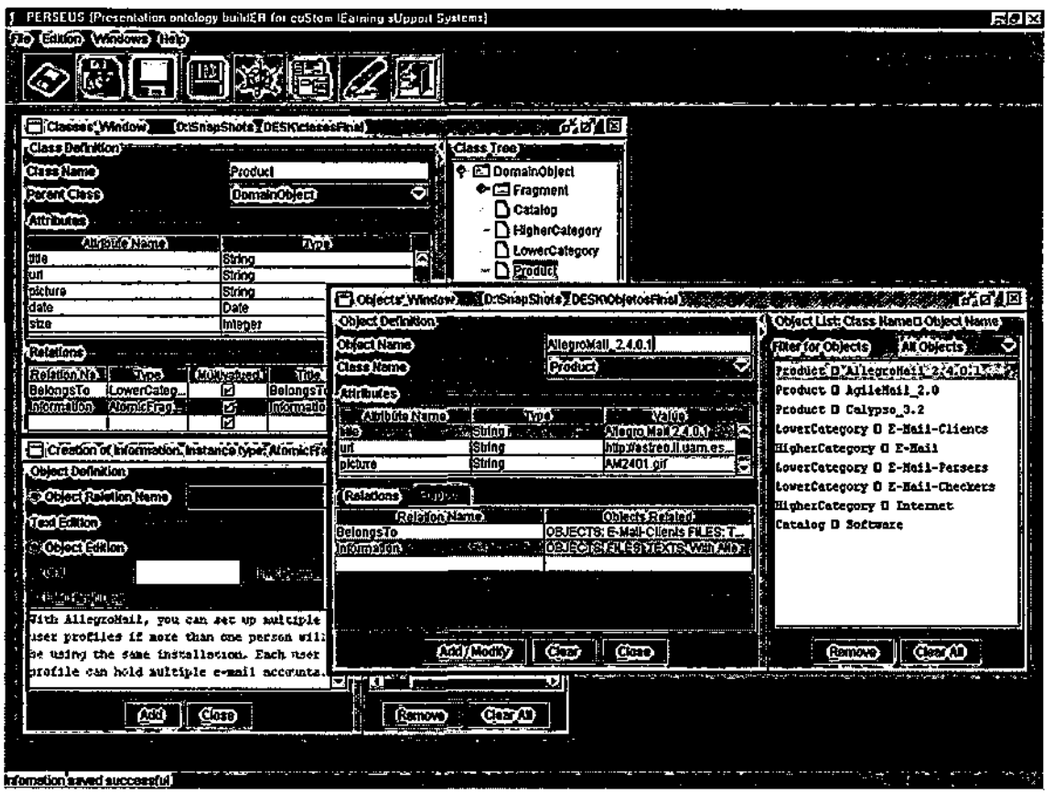


Figura 4.26: Diseño de la ontología y el modelo del dominio mediante PERSEUS

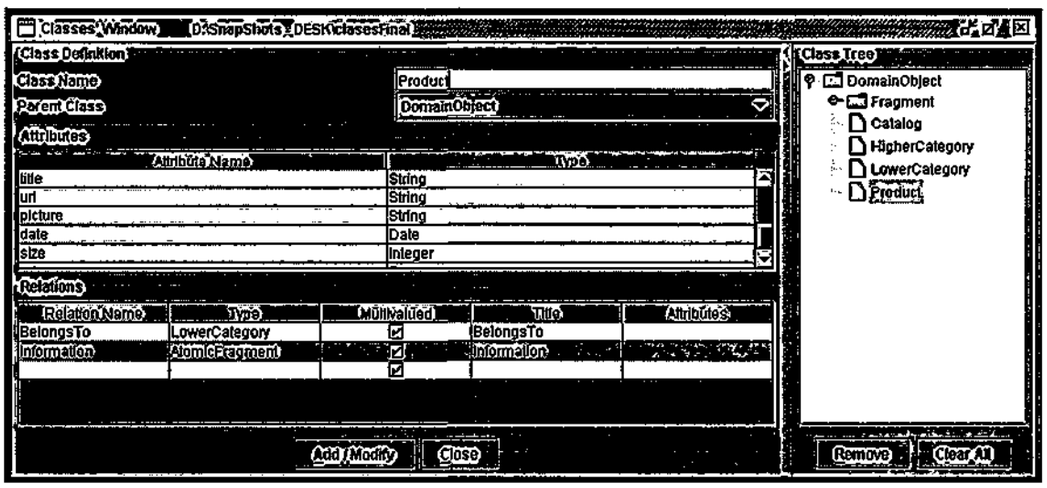


Figura 4.27: Definición de las clases que formarán la ontología del domino de la interfaz web

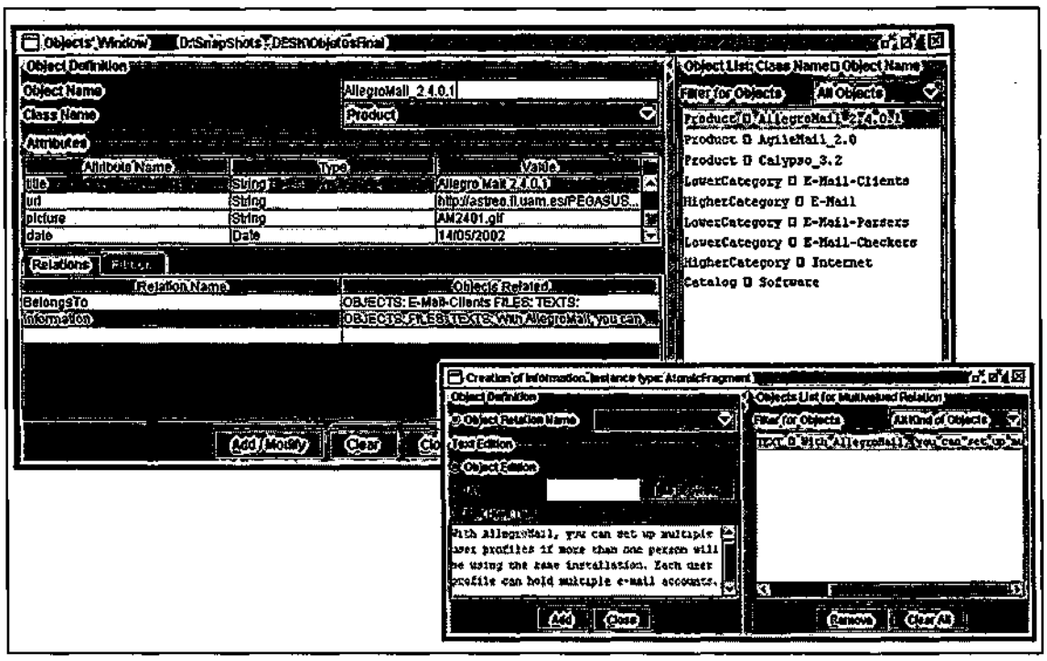


Figura 4.28: Instanciación de objetos del dominio (ventana superior) y de relaciones entre objetos (ventana inferior)

Una vez que se ha diseñado la ontología, el siguiente paso corresponde a la creación de la red de objetos del dominio (Figura 4.28), a partir de las clases creadas en la ontología (Figura 4.27). Para ello, teniendo en cuenta el ejemplo propuesto, se han creado distintas instancias, como los productos de correo electrónico `Allegro 2.4.0`, `AgileMail 2.0`, etc., pertenecientes a la categoría `E-Mail-Clients`, de tipo `LowerCategory`, pertenecientes a su vez a `E-Mail` de tipo `HigherCategory` e `Internet` como categoría superior, siendo el catálogo principal el objeto `Software`. Para cada uno de los objetos podemos definir sus relaciones, como es el caso de la relación multivaluada `Information` del objeto `AllegroMail` de la clase `Product`, que al tener una relación compuesta por objetos de tipo `AtomicFragment`, podemos además definir fragmentos multimedia para incorporar directamente en la presentación.

Finalmente, una vez que se ha completado y retocado el diseño tantas veces como sea necesario, el autor puede generar automáticamente el modelo del dominio completo simplemente pulsando sobre un botón de la herramienta PERSEUS, obteniendo así el XML necesario para ser procesado posteriormente por PEGASUS (Figura 4.29).

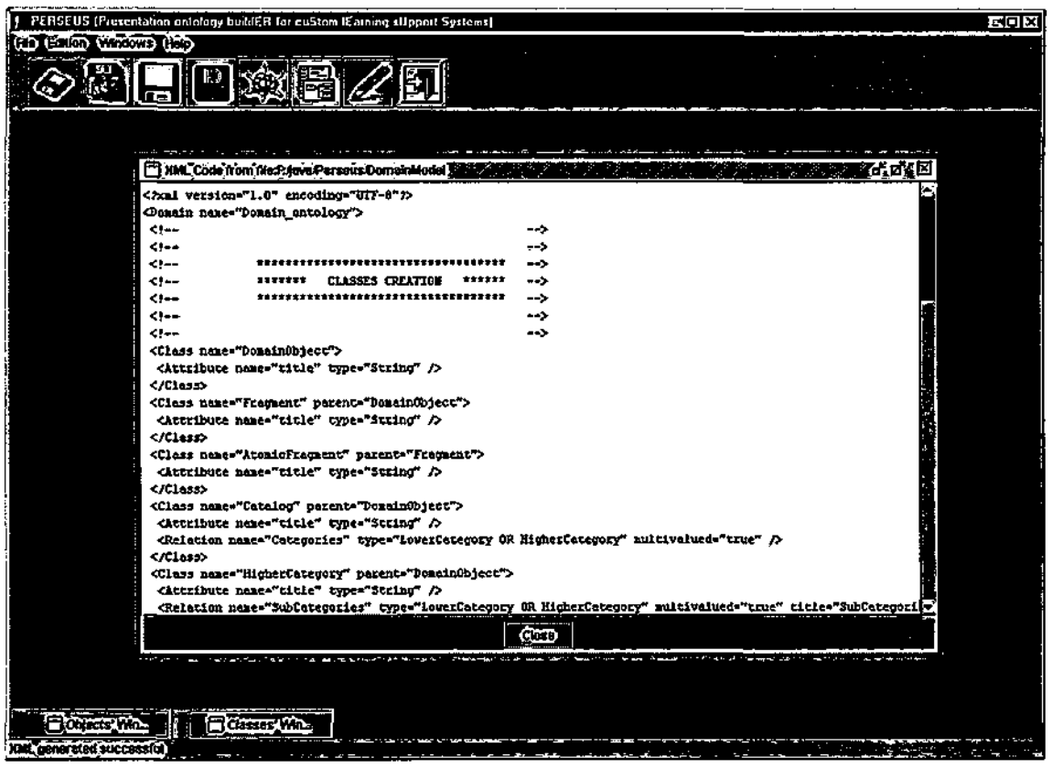


Figura 4.29: Modelo del dominio generado automáticamente por PERSEUS a partir de la ontología y los objetos del dominio creados por el autor

4.8.3 Generación de la presentación

Una vez que el modelo del dominio ha sido generado, un usuario autorizado (Figura 4.30) (i.e. un diseñador o un administrador) puede acceder, mediante el portal HADES, al envío de los datos al servidor utilizando cualquier navegador convencional. Para ello, solamente tiene que suministrar al portal el fichero XML o ZIP (si contiene, a parte del fichero XML, otros ficheros como imágenes, HTML adicional, etc.) en la página de creación de presentaciones web (Figura 4.31), respondiendo el sistema, una vez enviado el fichero, con información para el usuario sobre el resultado del proceso.

Una de las utilidades principales de HADES, en este caso, es la de generar automáticamente plantillas de presentación PEGASUS para visualizar la información relativa al modelo del dominio enviado por el diseñador. HADES permite generar estas plantillas a partir de una página donde el diseñador puede elegir modelos predefinidos, o incluso enviar un diseño propio mediante un sistema parecido al del envío del modelo del dominio comentado anteriormente. Para este ejemplo se han creado unas plantillas ad-hoc escritas a mano en el lenguaje de PEGASUS, en concreto una por clase, que posteriormente el usuario podrá refinar, adornar y añadirle diseños.

En la Figura 4.31 puede apreciarse cómo después de enviar el correspondiente fichero del modelo del dominio, el sistema responde aportando información sobre el número de clases, objetos y relaciones que componen el modelo procesado, así como el

éxito o la posibilidad de error en la *compilación* del código enviado por el diseñador al servidor.

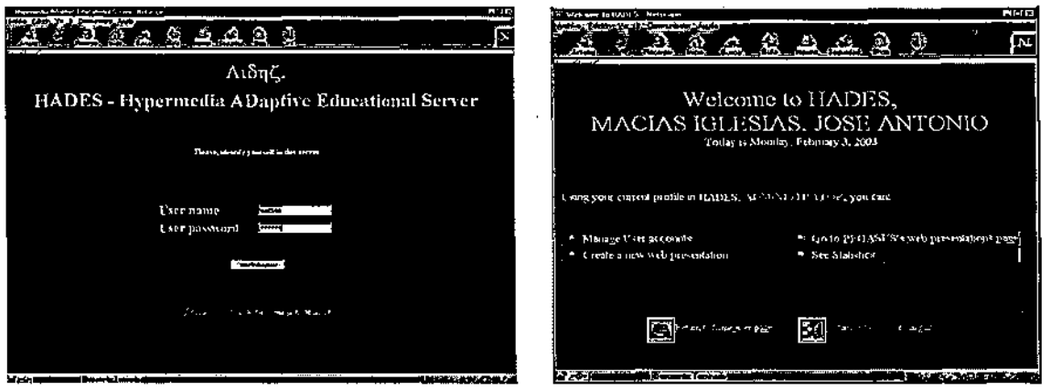


Figura 4.30: Entrada bajo auto-identificacion en el portal HADES (izquierda) y *home* del usuario, generado según el perfil correspondiente en el portal (derecha)

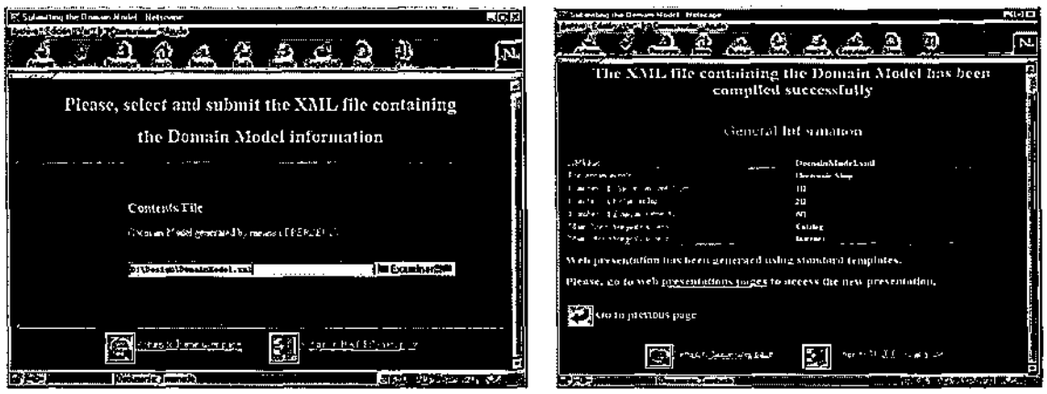


Figura 4.31: Pantalla de envío del modelo del dominio (izquierda) y pantalla de resultado sobre la *compilación* de la presentación y la generación de las plantillas PEGASUS (derecha)

Una vez que el proceso de generación ha finalizado con éxito, el usuario puede acceder, en función de su perfil, a las plantillas PEGASUS creadas en la página de HADES destinada a tal fin. En la Figura 4.32 se muestra la página de plantillas disponibles de PEGASUS, donde para el ejemplo propuesto aparecen todas las plantillas generadas (una por clase), junto con el acceso a los objetos instanciados para estas plantillas. Este acceso se lleva a cabo mediante la URL, como ya se explicó en apartados anteriores, de forma que cuando el usuario pulsa en el objeto elegido, el motor de generación de documentos web dinámicos de PEGASUS se pone en marcha, presentado al usuario el objeto seleccionado a partir de la plantilla. Esta plantilla se corresponde con el nombre de la clase a la que pertenece el propio objeto. A partir de entonces, el usuario interactúa con PEGASUS a través del portal HADES, controlando directamente PEGASUS la renderización de cada uno de los objetos del dominio a través de los mecanismos mencionados en el capítulo anterior.

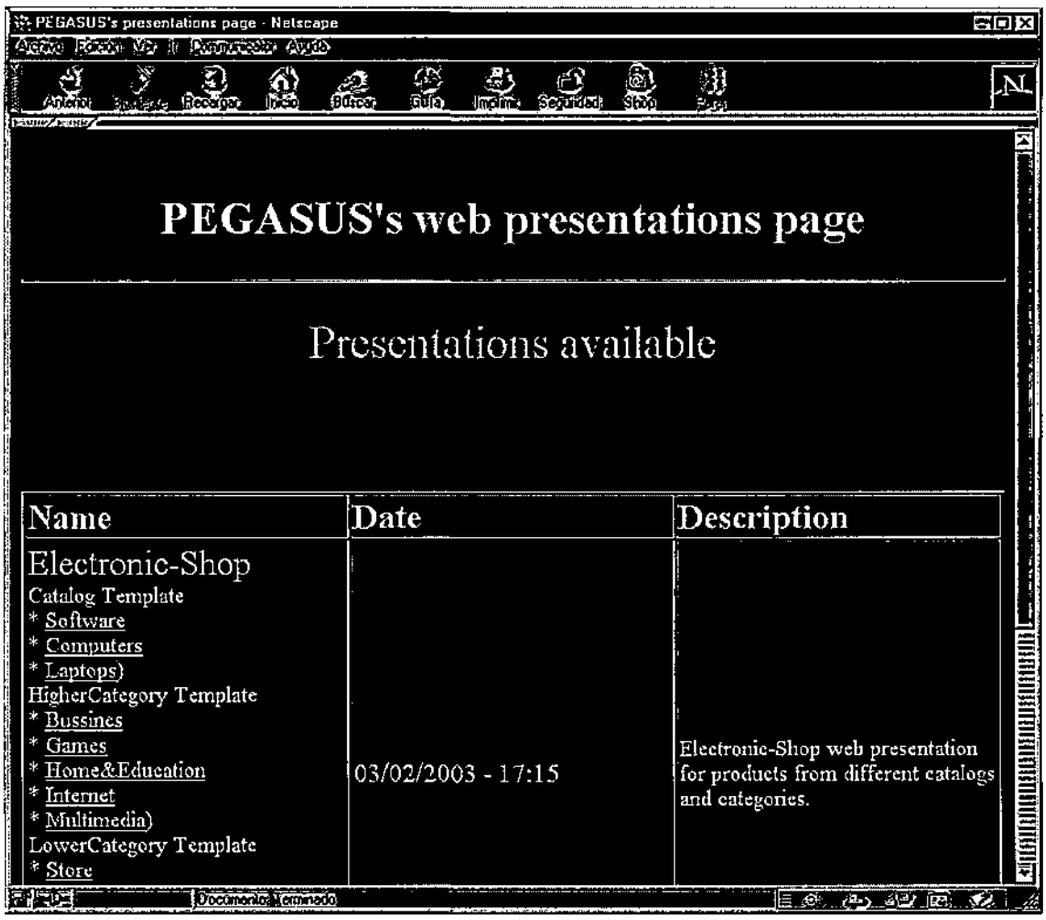


Figura 4.32: Página de presentación PEGASUS, donde puede verse la última presentación generada (Electronic-Shop) sobre la tienda electrónica elaborada como ejemplo

En la Figura 4.33 se muestra una página generada mediante PEGASUS para el objeto *Internet* mediante la plantilla *HigherCategory*, generada para renderizar o visualizar objetos de esa clase. El código correspondiente a esta plantilla, definida ad-hoc según se comentó en el párrafo anterior, es el que se muestra a continuación:

```
<widget type = "TabbedPane">
  <items> <%= parent-category.subcategories %> </items>
  <selectedItem> <%= ID %> </selectedItem>
</widget>

<%= title %>

<widget type="Table"
        columns="3"
        dataflow="wrap">
  <items> <%= subcategories %> </items>
</widget>
```

Esta plantilla está definida para representar los elementos de forma tabular a partir del widget tabla (`widget type="Table"`), estructurando jerárquicamente los elementos a partir de distintos niveles de categorías y sub-categorías. Además se utiliza

otro widget distinto (`widget type="TabbedPane"`) para mostrar, en la parte superior de la página web, el catálogo principal de productos. El widget de tipo tabla es un elemento predefinido del modelo de la presentación de PEGASUS (al igual que las reglas), y permite generar elementos complejos de la presentación, que se evalúan recursivamente en función de la relación del dominio a la cual van unidos (`<%= subcategories %>`). De esta forma es posible visualizar tablas anidadas recorriendo el grafo de la red semántica a partir de la relación del dominio establecida (`subcategories`). El elemento `<%= title %>`, por el contrario, permite visualizar el atributo *título* del objeto que se está renderizando mediante esta plantilla, de forma que en el caso de la Figura 4.33 será el atributo *título* del objeto *Internet*.

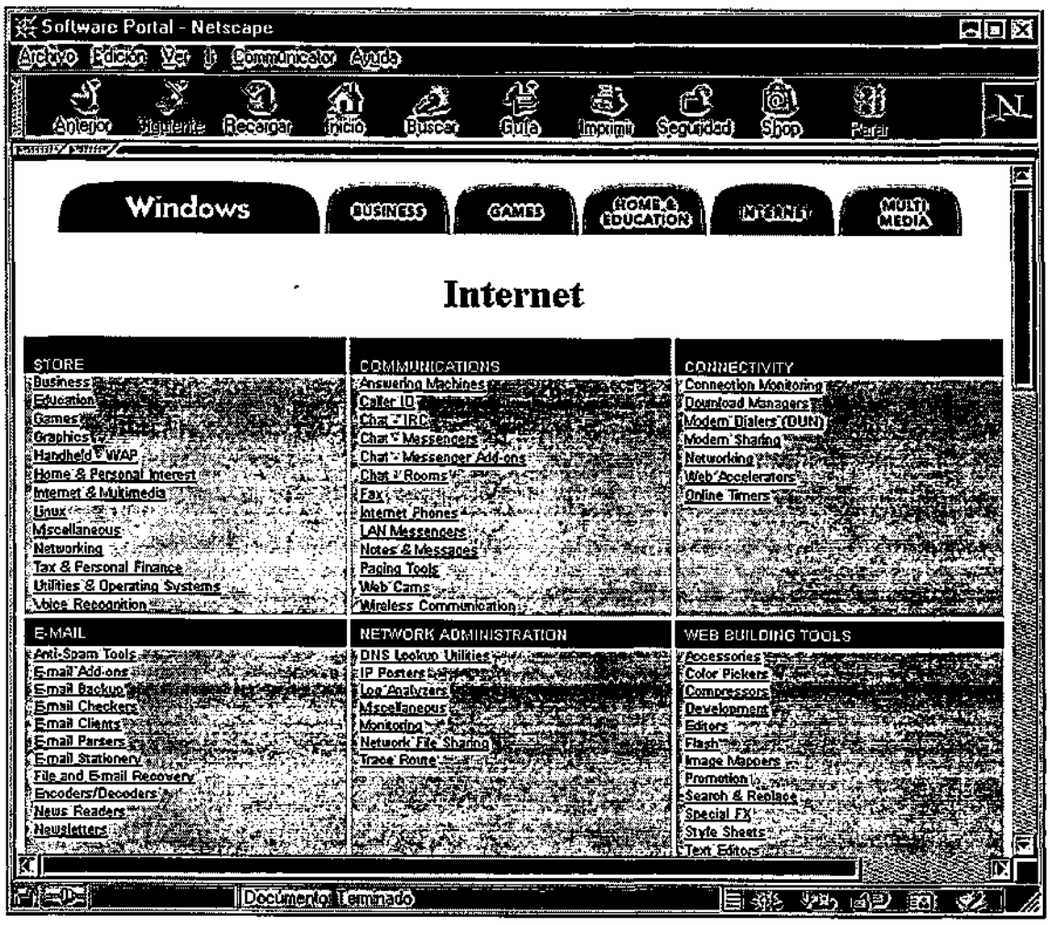


Figura 4.33: Presentación final completa generada por PEGASUS a partir del diseño llevado a cabo por el autor de la presentación

4.8.4 Autoría de la presentación generada

Una vez generadas las páginas web sobre la tienda electrónica, puede entrar en juego la autoría por parte de los usuarios que deseen realizar cambios en la apariencia o contenidos de la interfaz web. Para ello, y utilizando DESK como navegador, un usuario autorizado puede seleccionar mediante el portal HADES el objeto que quiere

visualizar. PEGASUS procesa la petición del usuario (i.e. el objeto solicitado) y genera código HTML que DESK interpreta y visualiza.

A partir de ese momento el usuario puede editar la página y efectuar cambios en la presentación, pudiendo solicitar un acceso a un objeto mediante HADES y navegando posteriormente por la presentación a partir de los links que PEGASUS genera dinámicamente.

En la Figura 4.34 se muestra un ejemplo de edición con DESK, teniendo en cuenta la presentación final de la Figura 4.33. En esta figura se representa cómo el usuario realiza los siguientes cambios:

a) Inserción del literal *Applications* justo después del literal *Internet*. Este cambio se lleva a cabo fácilmente ya que lo único que debe de hacer el usuario es situar el cursor en el lugar en el que quiere insertar la palabra y, seguidamente, escribirla. Como se verá, este cambio será interpretado por la herramienta como una adición de información al modelo del dominio. En este sentido DESK detecta el contexto más próximo (*Internet*), y procede a buscar el objeto correspondiente en el modelo del dominio para cambiar el valor del atributo (*título*) de este objeto, añadiéndole la nueva palabra y cambiando finalmente el valor inicial de este atributo por el de *Internet Applications*.

b) Inserción de dos widgets HTML: un combo box y una lista de selección. Para ello el usuario dispone de una barra de herramientas donde puede insertar y modificar widgets fácilmente. Este cambio producirá la inserción, en la plantilla de la presentación, de dos nuevos elementos de la presentación modelizados dentro de la plantilla como widgets. En este caso, la herramienta toma como contexto más cercanos los elementos de alrededor, es decir, la tabla con los productos y el panel del catálogo de la parte superior.

c) Copiar y pegar elementos entre controles HTML. Para copiar y pegar elementos dentro de algunos widgets en tiempo de diseño (como por ejemplo en un combo box o en una lista de selección), DESK añade unas ventanas especiales como las que se ven en la Figura 4.34, obtenidas a partir de hacer *doble-click* en el widget sobre el que se desea copiar o pegar elementos. De esta forma, en esta misma figura podemos ver cómo el usuario está copiando los elementos de la tabla de productos al combo box (`Store`, `Communications`, `Conectivity`, etc.), para las categorías de productos que aparecen en los encabezados de la tabla, y a la lista de selección (`Anti-Span Tools`, `E-mail Add-ons`, etc.), para las sub-categorías de productos que aparecen en cada celda de la tabla (p.e. la relativa a productos de tipo `E-mail`).

Otro hecho que se refleja en la Figura 4.34 es cómo después de copiar algunos elementos de un widget a otro el agente de inferencia DESK detecta la intención del usuario y le propone sustituir la tabla completa por un único combo box y una lista de selección (ver la ventana de diálogo central), los cuales contendrán las categorías y

subcategorías mostradas en la tabla, y que se corresponderán con los objetos que están siendo renderizados por PEGASUS en la instantánea de la edición perteneciente a esta página.

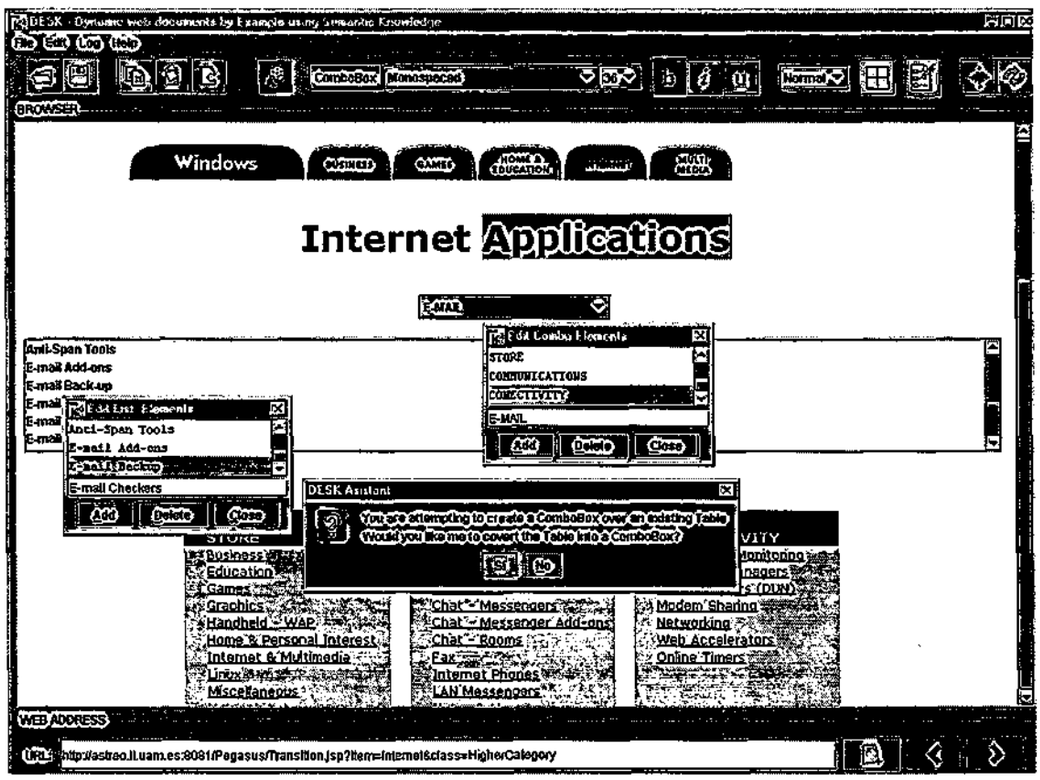


Figura 4.34: Autoría de la interfaz web bajo DESK

Las acciones realizadas por el usuario generan un modelo de monitorización representado en la Figura 4.35, donde puede apreciarse cómo el sistema identifica los cambios realizados por el usuario para luego inferir la información semántica correspondiente en la parte del servidor. En este ejemplo, el modelo de monitorización de la Figura 4.35 muestra dos primitivas, una donde el usuario ha añadido el literal *Applications* después del panel tabular (`after="TB01"`), antes de la tabla (`before="T01"`), y justo después del literal *Internet*, es decir:

```
...
  <inContext start_text=09 end_text=21 after="TB01" before="T01">
    Internet
  </inContext>
...
```

Otra de las primitivas añadidas representa la transformación de la tabla en un combo box y una lista de selección. Como puede comprobarse esta primitiva posee información semántica del dominio, debido a que ciertas estructuras de la interfaz (en este caso la tabla `T01`) son generadas mediante reglas. El sistema identifica las relaciones y objetos involucrados, y los utiliza posteriormente para mantener la correspondencia entre los objetos del dominio a partir de los posibles cambios que se

puedan dar entre widgets. También, en la primitiva `ID="11"` de la Figura 4.35, puede comprobarse cómo la herramienta genera la información relativa a los patrones de iteración detectados en la transformación (`ColShiftSet`, `RowShiftSet` y `ElemShiftSet`).

```
<Monitoring_Model>

  <InsertHTML ID="10">
    <TextAffected><![CDATA[Applications]]></TextAffected>
    <InContext starts="09" ends="21" after="TB01" before="T01">
      <![CDATA[Internet]]>
    </InContext>
  </InsertHTML>

  <ChangeWidget ID="11">
    <From type="Table" id="T01" relation_name="SubCategories"
          class="HigherCategory" objectID="Internet"
          ColShiftSet="(Col:1),1,…1" RowShiftSet="(Row:1),0,0…0"/>
    <To   type="ComboBox" id="C01" relation="SubCategories"/>
          class="HigherCategory
          ElemShiftSet="1"/>
    <To   type="List"  id="L01" relation="SubCategories"
          class="LowerCategory"
          ElemShiftSet="1"/>
  </ChangeWidget>

</Monitoring_Model>
```

Figura 4.35: Modelo de monitorización generado por DESK a partir de los cambio llevados a cabo por el usuario

La Figura 4.36 muestra un fragmento del código HTML generado por PEGASUS, el cual incluye meta-información correspondiente a la generación de la tabla a partir de un control HTML (la tabla, identificada bajo el sistema como `ID="T01"`), recogiendo las relaciones entre objetos del dominio (en negrita). La información embebida en este fragmento permitirá reconocer fácilmente el widget y su tipo (`widget_type="Table"`), junto con otros datos de interés que se tendrán en cuenta en el proceso de inferencia, como la relación del dominio a partir de la cual ha sido generado el widget (`relation_name="SubCategories"`) y que aporta información sobre los vínculos establecidos entre los modelos de la presentación y del dominio. Otros datos relevantes al respecto que se recogen son el nombre de la clase (`class_name`), el del objeto padre (`object_ID`) y el nombre del atributo que se muestra en cada celda (`title`). Dentro de la tabla se puede ver cómo, para cada uno de los sub-elementos o celdas que son a su vez tablas, se tiene nuevamente información embebida similar a la descrita anteriormente. De esta forma el mecanismo de inferencia

de DESK permite localizar más fácilmente qué objeto predefinido de la interfaz (widgets) ha dado lugar al fragmento de la Figura 4.36, pudiendo llevar a cabo DESK los cambios oportunos a partir de modificar valores en la configuración de estos elementos directamente en la plantilla o en los objetos del dominio embebidos.

```
<Table ID="T01" widget_type="Table" relation_name="SubCategories"
       class_name="HigherfCategory" object_ID="Internet">

  <tr> <td title="STORE">          <a href="..."> STORE          </a></td>
       <td title="COMMUNICATIONS"> <a href="..."> COMMUNICATIONS</a></td>
       <td title="CONNECTIVITY">   <a href="..."> CONNECTIVITY   </a></td></tr>

  <tr> <Table ID="T02" relation_name="SubCategories" class_name="LowerCategory"
              object_ID="STORE"
         <tr> <td attribute="title"> <a href="..."> Business   </a> <br>
                                     <a href="..."> Education  </a> <br>
                                     <a href="..."> Games      </a> <br>
                                                                  </td></tr>
              ...
       </Table> </tr>
  ...

  <tr> <td title="E-MAIL">          <a href="..."> E-MAIL       </a></td>
  ...                                                          </td></tr>

  ...
</Table>
```

Figura 4.36: Fragmento HTML con meta-información generada por PEGASUS sobre el widget tabla

En la Figura 4.37 se muestra un gráfico que representa la actuación del agente de inferencia DESK en función de los cambios llevados a cabo por el usuario en la presentación. Esta figura ilustra cómo a partir de la copia de elementos de un widget a otro el agente se activa y comprueba las precondiciones definidas en su configuración. Seguidamente activa las heurísticas que corresponden a los algoritmos de detección de patrones iterativos, completando así la secuencia iniciada por el usuario en la copia de elementos entre widgets, y manteniendo de forma automática las correspondencias semánticas entre objetos del dominio.

A partir del análisis del modelo de monitorización en el servidor, los cambios reflejados se llevan finalmente a cabo mediante la caracterización de cada uno de ellos. De esta forma, para el ejemplo mostrado en el Figura 4.35, estos cambios representan añadir el literal *Applications* (`ID="10"` en el modelo de monitorización) y la sustitución de la tabla por un combo box y una lista de selección (`ID="11"` en el modelo de monitorización).

Para llevar a cabo el cambio con `ID="10"` el sistema caracteriza el objeto del dominio afectado y el atributo donde se acomete el cambio, en este caso el atributo

`title` del objeto `Internet`, añadiendo la siguiente información semántica al modelo de monitorización para esa entrada:

```
<InsertHTML ID="10"
  ...
  <InContext>
    <FoundIn>
      <class="Category" ID="Internet" attribute="title">
    </FoundIn>
  </InContext>
</InsertHTML>
```

Cuando se procesa en el servidor esta información, se localiza el objeto del dominio a modificar (`Internet`) y se procede a añadir el literal "Applications" al título (atributo `title`) del mismo, resultando en "Internet Applications", como el usuario deseaba.

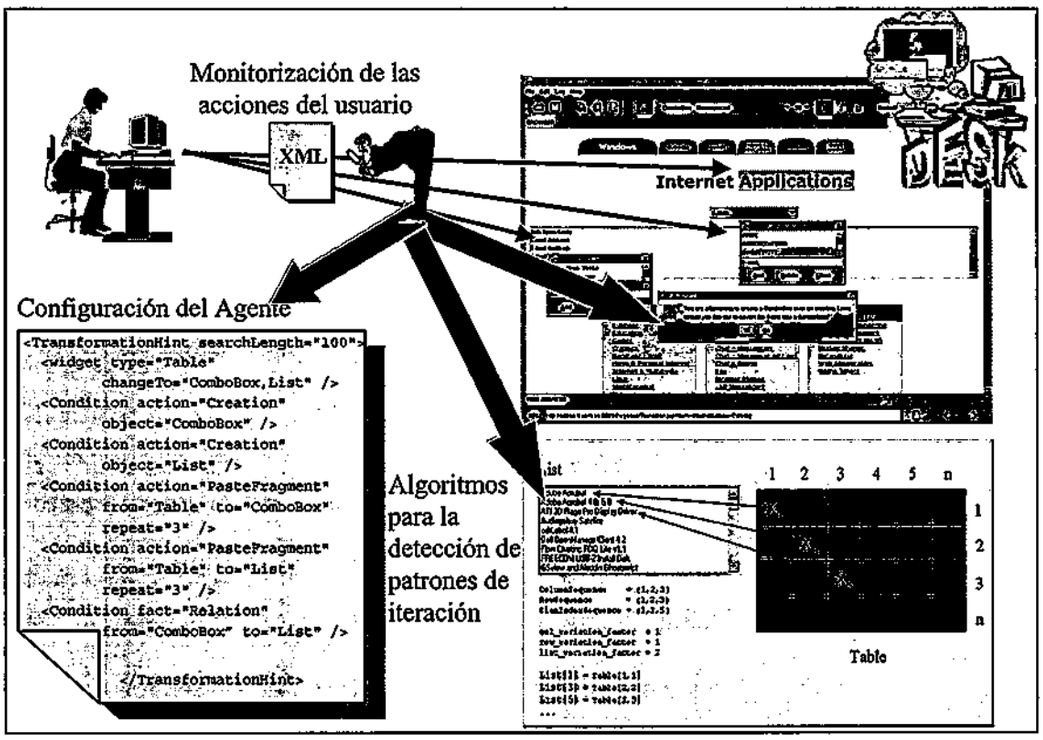


Figura 4.37: Actuación del agente a partir de los cambios llevados a cabo por el usuario en la edición de la presentación

Para el cambio con `ID="11"`, se procede a modificar la plantilla de la presentación que afecta al widget `Table`, es decir, al hecho de cambiar la tabla por un combo box y una lista de selección, conservando las mismas relaciones entre los objetos que existían originalmente (Figura 4.38). Este cambio resulta más sencillo de inferir, ya que como se vio anteriormente en la Figura 4.36 el sistema posee información semántica de partida, embebida en el código HTML final generado por PEGASUS y que DESK detecta en el cliente mientras navega. Otro factor que hace fácil el llevar a término este último cambio, es el hecho de modelizar los controles HTML de una forma

homogénea, lo cual facilita en gran medida realizar cambios de aspecto o transformaciones sin alterar su relación semántica con respecto al modelo del dominio (Figura 4.38).

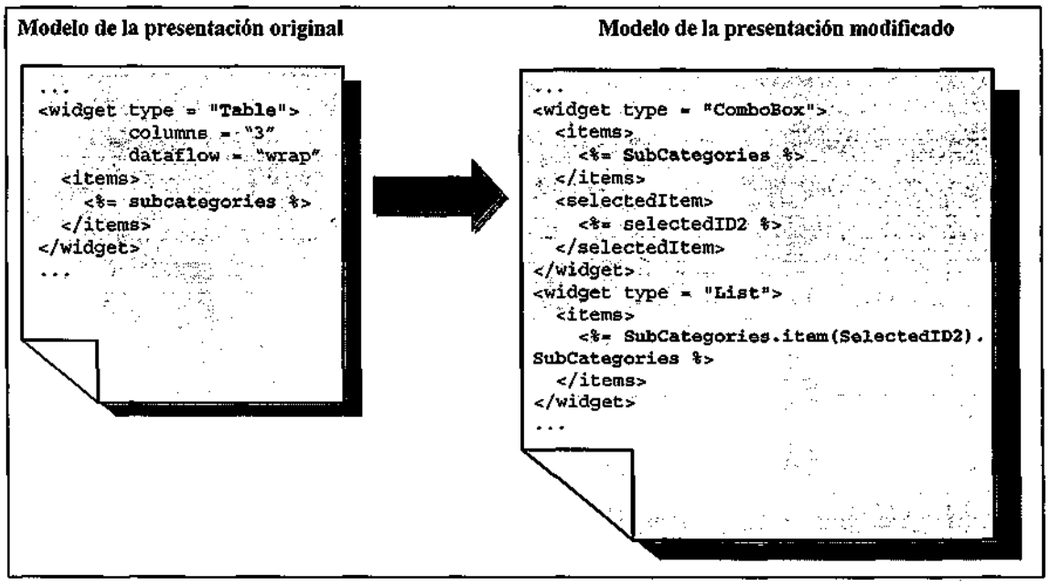


Figura 4.38: Fragmentos del modelo de la presentación indicando los cambios llevados a cabo por el sistema para transformar una tabla en una lista de selección y un combo box

El resultado final del proceso es el cambio en la apariencia de la presentación (Figura 4.39), es decir, se cambia la tabla por el combo box y la lista de selección, y se añade el literal *Applications*, tanto en la solapa principal de las categorías como en el texto del documento. En realidad se trata del mismo objeto que aparece doblemente en la presentación, de esta forma puede apreciarse como un cambio en un objeto puede afectar a todas sus coincidencias en el documento, reflejado internamente bajo PEGASUS como apariciones del mismo objeto dentro una plantilla de la presentación.

Como resumen, en este apartado se ha pretendido trasmitir, con este ejemplo de uso, el mecanismo seguido por la herramienta DESK para la caracterización de cambios que un usuario lleva a cabo en una página web generada por PEGASUS. El camino inverso seguido por DESK parte de los cambios que un usuario hace en una página HTML hasta llegar, mediante los mecanismos de inferencia narrados en este capítulo, a modificar la forma en la que se generarán las futuras páginas web por medio de PEGASUS, es decir, se modifican los modelos subyacentes a partir de los cuales se generan las páginas HTML por las que el usuario navega. Los mecanismos de inferencia vistos se apoyan en el modelo del dominio, como base para la caracterización de partes integrantes entre lo que el usuario edita y su correspondencia en la base de conocimiento. Por otro lado, también se han introducido otras herramientas de autor descritas en el capítulo anterior: PERSEUS, encargado de la autoría del contenido o modelo del dominio de PEGASUS, y HADES como portal de acceso y generación

automática de plantillas a partir de un diseño de la base de conocimiento creada por el diseñador.

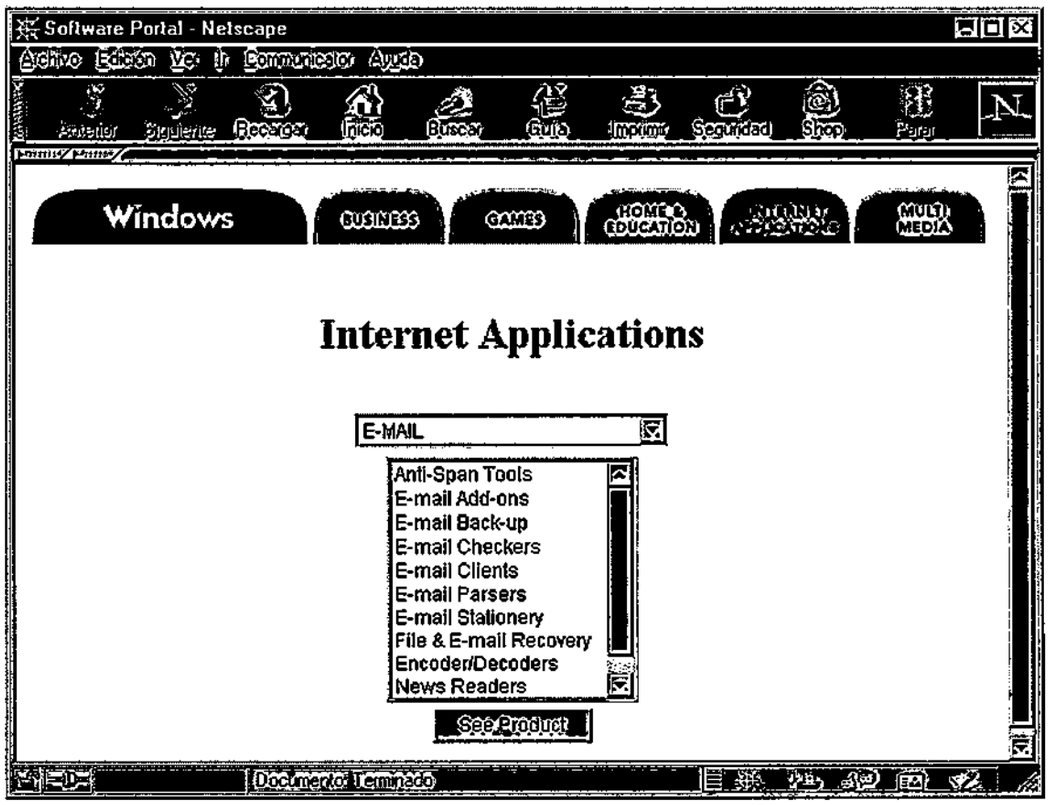
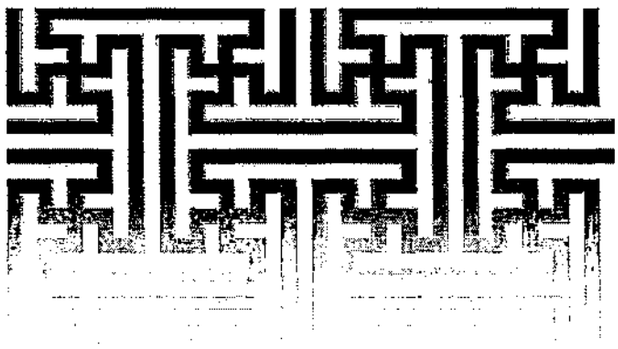


Figura 4.39: Presentación final después de los cambios llevados a cabo por el autor bajo la herramienta DESK



Capítulo V.
Evaluación y Conclusiones

En este capítulo se procederá a la evaluación del sistema propuesto para la autoría de páginas web dinámicas. Para ello se llevará a cabo un estudio empírico del sistema a partir de una experiencia realizada con usuarios. También se comparará DESK con otros sistemas o herramientas similares que hacen uso de la filosofía implícita en la solución propuesta. Así mismo se describirán las conclusiones y aportaciones principales, junto con las futuras líneas de trabajo que surgen como posibles ampliaciones a partir de lo expuesto en esta tesis.

5.1 Introducción

A través de los capítulos anteriores, se ha dado una visión detallada de los mecanismos empleados para la autoría de documentos web dinámicos aportados en esta tesis.

Por un lado, en el Capítulo I se pusieron de manifiesto los objetivos de esta tesis, proponiendo una problemática y acotando el problema a resolver. De esta forma se enunciaron los alcances y aportaciones de la tesis, así como una disección de la solución aportada a través de los distintos capítulos que componen este trabajo. Para contextualizar el trabajo de la tesis, así como la relación con las herramientas, sistemas y tecnología actual se dio, en el Capítulo II, un estado del arte sobre el trabajo relacionado, contando las bases de las principales técnicas y paradigmas empleados en el desarrollo de este trabajo, así como enumerando las principales aportaciones al respecto para seguir, en los siguientes capítulos, con las aportaciones principales de la tesis, consistentes en los mecanismos propuestos para la generación y autoría de documentos web dinámicos.

En el Capítulo III, se abordó la ventaja de separar contenidos y presentación, como vía para la edición independiente de cada uno de los elementos que componen un sistema de información hipermedia. Partiendo de esta idea ha sido posible abordar, en el Capítulo IV, la autoría de páginas web dinámicas por parte de un usuario final. A ese respecto, el mecanismo que se ha propuesto en esta tesis apunta a facilitar al usuario final la edición de una página HTML, generada dinámicamente por PEGASUS, haciendo especial hincapié en que éste es un problema propio del paradigma de la Programación por Demostración, que ayuda al usuario a abstraerse de los lenguajes de especificación utilizados para la generación de las propias páginas web que éste manipula. Adicionalmente, y como aportación relevante, se propuso la utilización de un modelo del dominio para caracterizar fragmentos *con sentido* dentro de la presentación, encontrando y analizando relaciones en la red semántica de objetos del dominio como técnica fundamental empleada, y aportando un asistente para realizar cambios de forma automática mediante la detección de patrones de iteración. Haciendo uso de este camino inverso, es decir, partiendo de una página concreta que el usuario modifica para llegar a los modelos que construyeron dicha página, DESK hace frente al problema de la edición dinámica de páginas web, modificando finalmente, de forma persistente, los procedimientos de generación de futuras páginas web que PEGASUS genera.

En primer lugar, en este último Capítulo V, se evaluará DESK como herramienta orientada al usuario final. Esta evaluación se llevará a cabo mediante la opinión de los usuarios finales sobre la herramienta, a partir de un caso de uso y obteniendo ciertos valores que serán tomados en cuenta para su discusión. Seguidamente se procederá a dar una comparativa y discusión sobre las herramientas más relacionados con DESK,

teniendo en cuenta lo narrado en el Capítulo II sobre trabajo relacionado, destacando los aspectos y aportaciones más importantes de la herramienta con respecto a las propuestas ya existentes. Seguidamente se darán unas conclusiones generales sobre el trabajo llevado a cabo a lo largo de esta tesis para terminar, finalmente, con el trabajo futuro que da pie a la continuidad de la investigación.

5.2 Evaluación empírica del sistema propuesto

Para validar la herramienta de autor DESK, se ha llevado a cabo una experiencia que permitirá contrastar la opinión de usuarios finales sobre DESK en lo referente a la edición de documentos web, evaluando la *usabilidad* de la herramienta en sí y el índice de acierto o fallo a partir de una serie de modificaciones que los usuarios llevan a cabo mediante DESK.

Para elaborar el ejemplo propuesto en la experiencia se han utilizado cada una de las herramientas vistas en capítulos anteriores, las cuales comprenden el sistema de generación y autoría de documentos web expuesto en esta tesis. En cualquier caso, se ha estimado oportuno realizar la evaluación no de todas las herramientas, sino sólamente de la herramienta de autor destinada a la interacción directa con el usuario final, es decir, la herramienta de autor DESK que es el eje de esta tesis. En cualquier caso las páginas web finales serán generadas con PEGASUS, existiendo una estrecha relación entre ambas herramientas, si bien el usuario no necesita saber nada adicional sobre PEGASUS para completar la experiencia que se propone.

5.2.1 Experiencia con DESK

La experiencia se llevó cabo a cabo teniendo en cuenta las siguientes premisas:

1) Tipo de usuario al que va orientada la experiencia. La experiencia propuesta fue pensada para usuarios que tuvieran cierto manejo con herramientas de edición para la web, o que tuvieran suficiente capacidad de abstracción sobre la forma de trabajar en este tipo de entorno. De esta forma se procedió a instruir previamente a los usuarios durante 10 minutos, previos a la experiencia, además de suministrarles con anterioridad material escrito sobre la herramienta en la que iban a trabajar. El adiestramiento mencionado consistió básicamente en una sencilla explicación sobre el formalismo seguido por la herramienta y la metodología utilizada para llevar a término la experiencia.

2) Objetivos principales. Los objetivos principales que persigue esta experiencia son los siguientes:

    a. Evaluar la sencillez de manejo de la herramienta.

b. Comprobar que los usuarios se hacen fácilmente con el mecanismo empleado para la autoría de páginas web dinámicas.

c. Observar si la herramienta les recuerda o les resulta familiar con respecto a uso de otras herramientas de autor parecidas, como por ejemplo las herramientas para la edición de páginas web estáticas.

d. Detectar si el usuario se siente en algún momento frustrado a partir de algo que esperaba poder hacer. Es decir, se trata de saber si hay algo que no ha conseguido porque le ha resultado difícil o porque la herramienta no le ha proporcionado los medios para llevarlo a cabo.

e. Evaluar el índice de aciertos, fallos y/o ambigüedades en la inferencia de los cambios bajo la herramienta, a partir de las acciones llevadas a cabo por los usuarios.

3) Metodología empleada. La metodología empleada para llevar a cabo la experiencia se basa, inicialmente, en el adiestramiento previo del usuario durante un corto espacio de tiempo. Seguidamente se le muestra al usuario un diseño final (en papel), al cual tiene que llegar mediante una página web inicial generada dinámicamente por PEGASUS, y en la que tendrá que realizar los cambios oportunos para llegar al resultado final que se le ha mostrado. Trascurrido ese tiempo se obtienen valores cuantitativos sobre la interacción, a partir de los ficheros de *log* generados por la herramienta, y se le proporciona al usuario un cuestionario a completar, del cual se extraerá información muy concreta sobre aspectos relacionados con la visión del usuario final sobre DESK.

4) Métricas utilizadas en la evaluación. Los aspectos cuantitativos más relevantes a estudiar en la interacción del usuario con la herramienta son los siguientes:

a. Tiempo utilizado por el usuario para llevar a cabo los cambios propuestos.

b. Secuencialidad del usuario en la ejecución de los cambios y cómo esto afecta a la inferencia.

c. Número de primitivas generadas por el usuario en la interacción (número de acciones en el modelo de monitorización)

d. Número de primitivas cuya información semántica ha sido localizada correctamente, es decir, modificaciones que finalmente se han llevado a término de forma exitosa por DESK.

e. Número de ambigüedades producidas, así como el estudio y resolución de las mismas por parte de la herramienta.

f. Índice de acierto o fallo de la herramienta para cada cambio propuesto, teniendo en cuenta los errores generados y su causa.

5) Aspectos cualitativos utilizados en la evaluación. Además del aspecto cuantitativo, el usuario es evaluado a partir de un cuestionario (mirar el Anexo de esta tesis) donde se estudian aspectos relativos a la *usabilidad* de la herramienta, es decir, lo fácil, difícil, útil y apropiada que le ha resultado al usuario la herramienta a la hora de llevar a cabo la experiencia propuesta, así como posible deficiencias percibidas y donde se pondrán en juego los aspectos en los que DESK puede resultar más apropiado o más propenso a fallos.

5.2.2 El ejemplo propuesto

El conocimiento de la presentación fue creado mediante PERSEUS, siendo la interfaz web elegida generada posteriormente por PEGASUS, a partir de un modelo del dominio relacionado con el mundo del submarinismo. De esta forma se construyó información del dominio consistente en una ontología definida a partir de unas pocas clases (`Topic`, `Fragment`, `AtomicFragment`, `Concept`, y `Content`) con relaciones muy simples (`prerequisites`, `procedure`, `explanation`, y `outline`), como se muestra a continuación:

```
...
<Class name="Item"/>
<Class name="Topic" parent="Item">
      <Relation name="subtopics"      title="Subtopics"/>
      <Relation name="prerequisites" title="Conceptos Previos"/>
</Class>
<Class name="Fragment"         parent="Item"/>
<Class name="AtomicFragment"  parent="Fragment" />

<Class name="Course" parent="Topic">
      <Relation name="items"/>
</Class>


<Class name="Concept" parent="Topic">
      <Relation name="procedure" title="Concepto"/>
      <Relation name="Outline"    title="Puntos a tener en cuenta"
type="Content"/>
</Class>

<Class name="Content" parent="Fragment">
      <Relation name="Explanation" title="Explicacion"
multivalued="no"/>
</Class>
```

Estas clases permiten instanciar objetos para mostrar un curso de submarinismo, creando finalmente algunos de esos objetos, como el concepto C5 consistente en una explicación relativa a la *Planificación y Control de las Inmersiones*, tal y como se muestra a continuación:

```
<Course ID="BIE" TITLE="Curso de Submarinismo">
  <Items>
    <Concept ID="C5" TITLE="Planificacion y Control de las
Inmersiones" read="yes">
      <prerequisites>
        <Concept REF="C1"/>
        <Concept REF="C2"/>
        <Concept REF="C3"/>
        <Concept REF="C4"/>
      </prerequisites>
      <procedure>
        <AtomicFragment>
          <![CDATA[En este capitulo conoceremos como se puede bucear
alejado del riesgo de la sobresaturacion oritica <br> de nitrogeno, lo
que llamamos bucear sin rebasar la curva de seguridad. <br><br>
                  Conocer que es la curva de seguridad.<br>
                  Conocer a que llamamos tiempo en el fondo<br>
                  Conocer a que llamamos tiempo limite<br>
                  Conocer a que llamamos profundidad de la
inmersion<br>
                  Conocer a que llamamos frontera de seguridad<br>
                  Conocer cuales son los factores que incrementan el
riesgo de la E.D.<br>]]>
        </AtomicFragment>
        <AtomicFragment URL="Inmersiones.jsp"/>
        </procedure>
        <Outline>
          <Content TITLE="Inmersiones Simples">
            <Explanation>
              <AtomicFragment>
                <![CDATA[Son aquellas en las que la ultima inmersion se
llevo a cabo en mas de 12 horas. <br> Actuaremos normalmente, como si
se tratara de una primera inmersion]]>
              </AtomicFragment>
              </Explanation>
          </Content>
          <Content REF="IC"/>
          <Content TITLE="Inmersiones Sucesivas">
            <Explanation>
              <AtomicFragment>
                <![CDATA[Aquellas en las que la ultima inmersion se
llevo a cabo en mas de 10 minutos pero menos de 12 horas de la
anterior<br><br>
                                  El tiempo limite de una inmersion
sucesiva es tl = TL-TNR <BR>
                                  Para calcular este tiempo limite (tl)
tendremos
                                  que calcular el tiempo limite de una
inmersion sencilla (TL) a esa profundidad <br>
                                   (Mediante la Tabla del Tiempo Limite y
Coeficientes de Salida) y restarle el tiempo de nitrogeno residual
(TNR)]]>
                              </AtomicFragment>
                            </Explanation>
                          </Content>
                          <Content TITLE="Tablas de Inmersion" read="no">
                            <statement>
                              <AtomicFragment URL="Tablas.jsp"/>
                            </statement>
                          </Content>
        </Outline>
```

```
    </Concept>
%>
```

Para renderizar esta información se generaron, mediante HADES, unas plantillas de presentación para las clases `Concept` y `Content`, las cuales muestran información muy básica sobre los objetos pertenecientes a ambas clases, así como las relaciones involucradas en cada uno de ellos. Ambas plantillas tienen un aspecto muy similar, parecido al que se muestra a continuación:

```
%>
<h3>  <%= prerequisites.title (0) %> </h3>
      <%= prerequisites.tree (0) %>
<h2>  <%= title %>                   </h2>
<h3>  <%= procedure.title (0) %>     </h3>
      <%= procedure %>
<h3>  <%= outline.title (0) %>       </h3>
      <%= outline %>
```

Utilizando PEGASUS se generó la interfaz web a partir de los datos del dominio y de las plantillas de presentación, mostrándose al usuario, mediante DESK, la página inicial a utilizar para llevar a cabo los cambios establecidos (Figura 5.1). Estos cambios fueron ejecutados e impresos previamente en papel para mostrar al usuario cómo debía quedar la presentación después de llevar a cabo dichos cambios. En el Anexo de esta tesis se pueden ver las páginas inicial y final utilizadas durante la experiencia. Respecto a la página final, está basada en los siguientes cambios:

a) Cambiar el literal "Planificación y Control de las Inmersiones" por "Planificación, Control y Estudio de las Inmersiones".
b) Subrayar el literal "...bucear alejado del riesgo de la sobresaturación critica...".
c) Crear una tabla centrada, de dos filas y una columna, inmediatamente antes del literal "Puntos a tener en cuenta".
d) Insertar en la fila número 1 el literal "OBJETIVOS".
e) Copiar la lista de literales "Conocer que es la curva de seguridad ... Conocer cuales son los factores ..." a la fila número 2 de la tabla (cut) borrándolo de su localización anterior (paste).
f) Poner en cursiva la lista de literales "Conocer que es la curva de seguridad ... Conocer cuales son los factores ...".
g) Crear un lista de *bullets* para la lista de literales "Conocer que es la curva de seguridad ... Conocer cuales son los factores ...", para indicar cada uno de los puntos establecidos:
    i. Conocer que es la curva de seguridad.
    ii. Conocer a que llamamos tiempo en el fondo
    iii. Conocer a que llamamos tiempo limite
    iv. Conocer a que llamamos profundidad de la inmersión
    v. Conocer a que llamamos frontera de seguridad
    vi. Conocer cuales son los factores que incrementan el riesgo de la E.D.

h) Cambiar, en el literal: " En este caso, la segunda inmersión se considera como continuación de la primera", por este otro: "En este tipo de inmersiones, la inmersión que se ha considerado como segunda en la Figura de arriba se convertirá automáticamente en la continuación de la primera".
i) Cambiar el estilo del literal "Inmersiones Simples" y ponerlo en H2.
j) Cambiar el estilo del literal "tl = TL-TNR" y ponerlo en negrita.

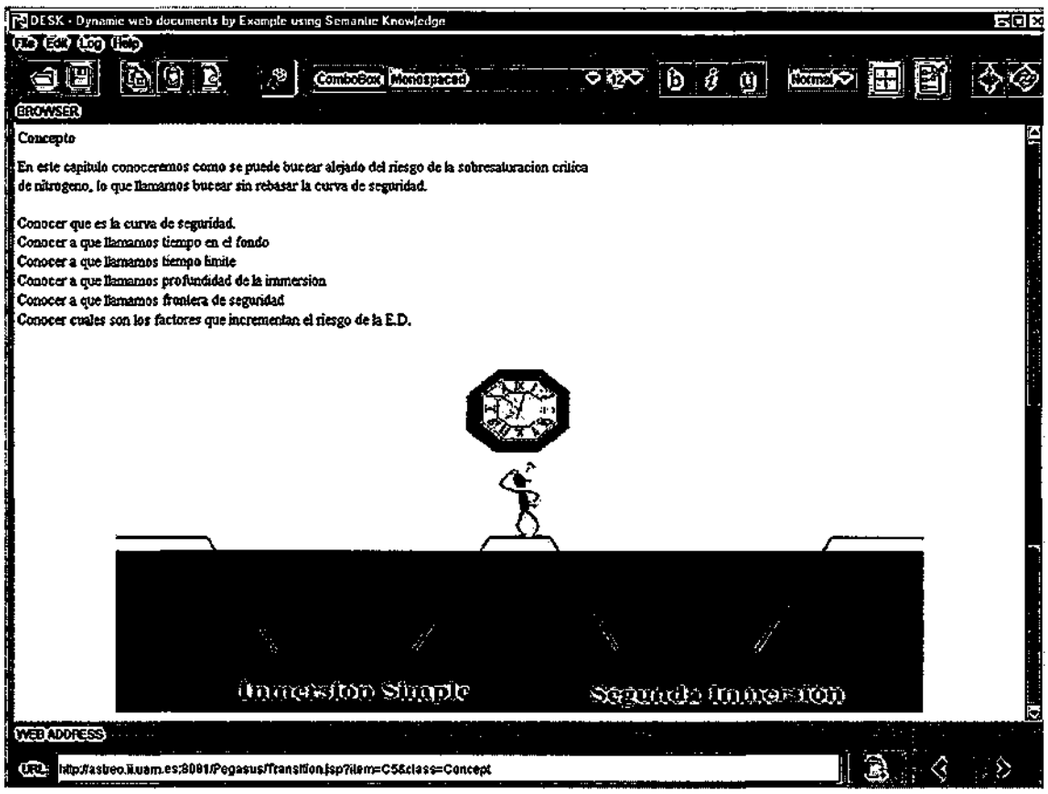


Figura 5.1: Instantánea tomada de DESK, a partir de la presentación generada por PEGASUS mediante las plantillas de la presentación y el modelo del dominio sobre submarinismo

5.2.3 Resultados, discusión y conclusiones sobre la experiencia

La experiencia descrita se llevó a cabo finalmente con 12 usuarios seleccionados en un entorno cercano. De la interacción directa con el usuario bajo la herramienta se extrajeron una serie de datos que serán analizados posteriormente. Así mismo se procedió a presentar al usuario, al final de la experiencia, un cuestionario destinado a obtener la opinión de éste sobre ciertos aspectos de la herramienta. Estos datos, igualmente, serán analizados en los próximos apartados.

5.2.3.1 Datos medidos durante la interacción

Los usuarios que participaron en la experiencia, tal y como demostró el test final, ya estaban adiestrados con distintas herramientas de edición de páginas estáticas para la web, como FrontPage®, Composer®, Dreamweaver®, etc. Por lo tanto, los requisitos iniciales estaban cumplidos con creces respecto a las exigencias expuestas para llevar a cabo la experiencia.

Durante la experiencia se controló en todo momento el tiempo utilizado por los usuarios para llevar a cabo las modificaciones propuestas. Este resultado se muestra en la Tabla 5.1, donde se ha tomado como referencia el tiempo máximo, mínimo y medio, así como la desviación observada. A partir de estos valores se comprueba que el tiempo para llevar a cabo la experiencia por parte de los usuarios es, en general, relativamente bajo. Esto concuerda con la hipótesis de que los usuarios estaban ya adiestrados con herramientas similares y que, por tanto, DESK les recordó en cierta medida la idea abstracta de un entorno de edición WYSIWYG como los comentados anteriormente. Todos los usuarios consiguieron acabar la experiencia de forma exitosa.

	Tiempo
Máximo	**7 minutos**
Mínimo	**4 minutos y 11 segundos**
Media	**5 minutos y 39 segundos**
Desviación	**0.8**

Tabla 5.1: Tiempo utilizado por los usuarios para completar la experiencia

Por otro lado, se midió si el usuario siguió la secuencialidad implícita en la página final mostrada para llevar a cabo los cambios propuestos, es decir, si seguían los cambios desde el comienzo hasta el final de la página según éstos iban apareciendo de forma secuencial en la hoja que se suministraba. Este resultado afecta a la generación del modelo del usuario, en la medida en que el contexto para un determinado cambio puede variar o verse afectado por lo que el usuario ha realizado anteriormente. En general este factor es controlado automáticamente por DESK, garantizando la no aparición de inconsistencias en el modelo y controlando, de forma robusta, los cambios de contexto independientemente del orden en el que se realicen dichos cambios. Atendiendo a este factor, el usuario puede elegir el efectuar los cambios como desee, aprovechando incluso los elementos ya cambiados o que han aparecido en acciones anteriores, y dando esto a un número distinto de primitivas generadas en el modelo de monitorización final a ser procesado en el servidor. En la Figura 5.2 se muestra, en cualquier caso, como el usuario prefirió realizar los cambios, mayormente, de forma secuencial.

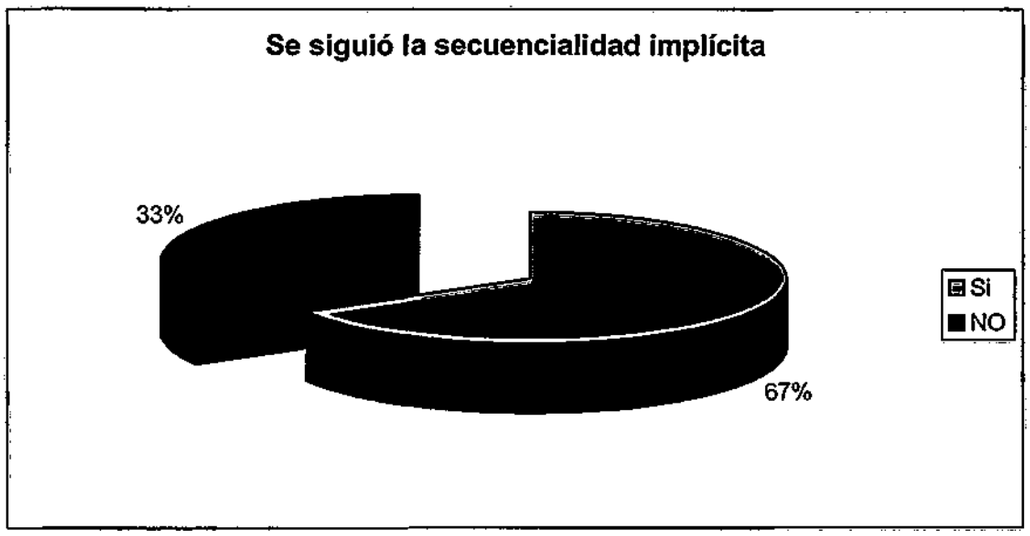


Figura 5.2: Porcentaje de usuarios que decidieron, o no, seguir la secuencia de cambios reflejados implícitamente en la presentación final

Otra medida interesante a tener en cuenta, siguiendo con lo comentado en el párrafo anterior, es el número de primitivas generadas por el usuario durante la interacción, el cual se corresponde con las acciones de edición llevadas a cabo por el usuario y que, posteriormente, formarán el modelo de monitorización que se enviará al servidor. En la Tabla 5.2 se muestra la estadística de primitivas utilizadas. Esto demuestra que cada usuario llevó a cabo un número distinto de acciones para completar la experiencia, como se observa en el valor de la desviación, dependiendo del orden y de la forma de llevar a cabo cada uno de los cambio bajo el entorno de edición.

	N° Primitivas
Máximo	281
Mínimo	155
Media	200
Desviación	34

Tabla 5.2: Número de acciones realizadas por el usuario que llevaron a la construcción del modelo de monitorización enviado al servidor

A partir del número de primitivas generadas, se puede obtener un dato relevante una vez enviado el modelo de monitorización al servidor, el cual permitirá saber el índice de acierto o fallo de la herramienta. Este dato es el número de primitivas a las cuales se les ha asignado contexto semántico, y que coincide con el número de acciones acometidas en la parte del servidor. Como puede verse, este valor es relativamente alto, en torno al 95% de media. El resto de primitivas que no han sido acometidas fueron casos muy concretos de saltos de carro, y algunas operaciones confusas entre bloques que en su mayoría produjeron cierta ambigüedad, en el sentido en que la herramienta no

supo asignarles un contexto semántico adecuado. Por otro lado, el medir el índice de acierto sobre el número de primitivas es más apropiado que el hacerlo sobre el número de cambios (a alto nivel) propuestos al usuario para ser llevados a cabo, debido a que este valor en la totalidad de los casos ha sido próximo en media al 98%. Sin embargo, y como ya se dijo anteriormente, dependiendo de cómo haya realizado el usuario estos cambios se generará un conjunto distinto de primitivas en cada caso, siento por tanto más robusto el analizar el número de primitivas bien acometidas, en función de las acciones de bajo nivel llevadas a cabo por el usuario en el entorno de edición DESK. Este valor se muestra en la Tabla 5.3, donde puede verse como en media la herramienta ha reaccionado como se esperaba.

	Porcentaje de Acierto
Máximo	98%
Mínimo	90%
Media	95%
Desviación	2.9

Tabla 5.3: Porcentaje de acierto de la herramienta a partir del número de primitivas acometidas en el servidor

5.2.3.2 Opinión de los usuarios sobre DESK

Utilizando el cuestionario, y en relación con lo estudiado en el apartado anterior, se le preguntó al usuario por el grado de predicibilidad de la herramienta, en términos de valores entre 5 (muy predecible) y 0 (poco predecible). Esta característica afecta al hecho de cómo el usuario percibe que la herramienta ha intuido o inferido los cambios llevados a cabo, y en qué grado DESK los ha hecho efectivo. La estadística muestra (Tabla 5.4) como realmente los usuarios piensan en media que la herramienta ha satisfecho sus cambios con un alto grado de predicibilidad, salvo en los casos comentados anteriormente y que tienen que ver con algunas acciones del modelo de monitorización que no pudieron ser llevadas a término.

	Grado de Predecibilidad (0-5)
Máximo	5
Mínimo	3
Media	4.2
Desviación	0.6

Tabla 5.4: Grado de predecibilidad de la herramienta a partir de la opinión de los usuarios; valores entre 5 (máximo) y 0 (mínimo)

Teniendo en cuenta este último valor y, a partir de algunos de los errores estudiados en el porcentaje de acierto de la herramienta, se les ha preguntado a los usuarios por la percepción que hayan podido tener de los fallos durante el proceso de

edición así como después de haber enviado los cambios al servidor (Figura 5.3). En este sentido, y como se comentó antes, el mayor índice de percepción de fallos viene dado a partir del tratamiento de algunos bloques y controles HTML, como tablas y listas, que han sido en ciertos momentos difíciles de localizar o de editar dentro de la herramienta. Este tipo de fallos merece una especial atención, así como un futuro esfuerzo por mejorar estos errores relativos al manejo de estructuras complejas.

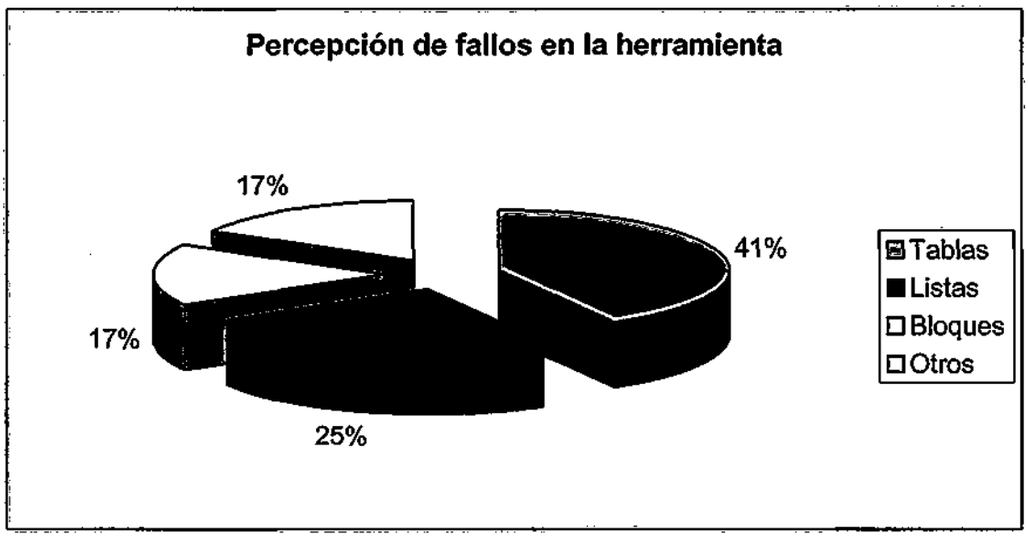


Figura 5.3: Percepción de fallos en la herramienta por parte de los usuarios en la manipulación de ciertos elementos de la presentación

A partir de la encuesta realizada a los usuarios se han obtenido distintas conclusiones que afectan directamente a la usabilidad de la herramienta en sí. Uno de los factores medidos ha sido la facilidad de uso de la herramienta por parte de los usuarios que, como puede comprobarse en la Figura 5.4, ha dado como resultado el que los usuarios opinan en mayoría que la herramienta es en sí fácil de utilizar.

Por otro lado, la dificultad utilizada en comprender el formalismo llevado a cabo por la herramienta ha sido, en su mayoría, superada por los usuarios sin problemas, tal y como se describe en la Figura 5.5, donde se muestra cómo en la mayoría de los casos los usuarios han comprendido perfectamente el fin de la herramienta, habiendo abstraído perfectamente la metodología de trabajo con ella.

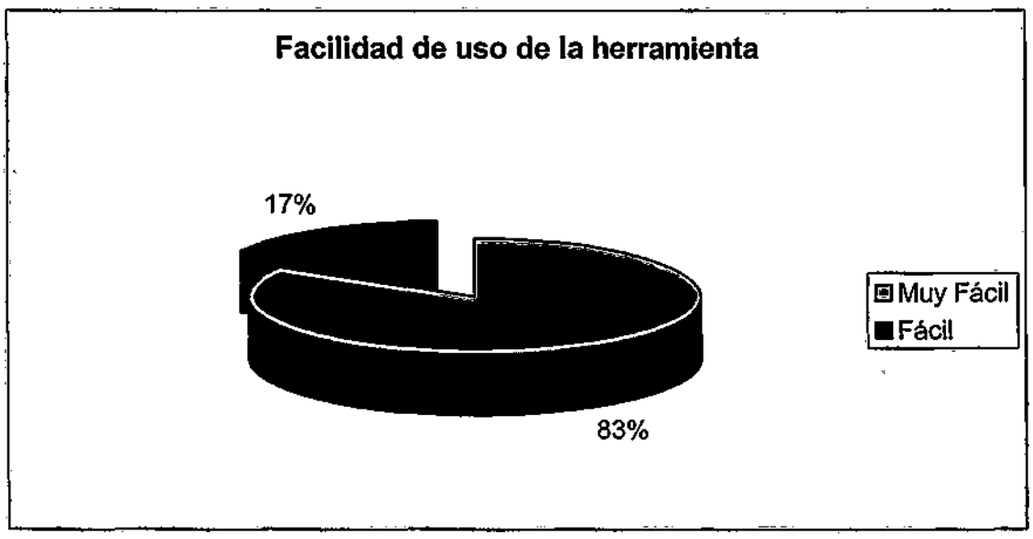


Figura 5.4: Facilidad de uso de la herramienta a partir de la opinión de los usuarios

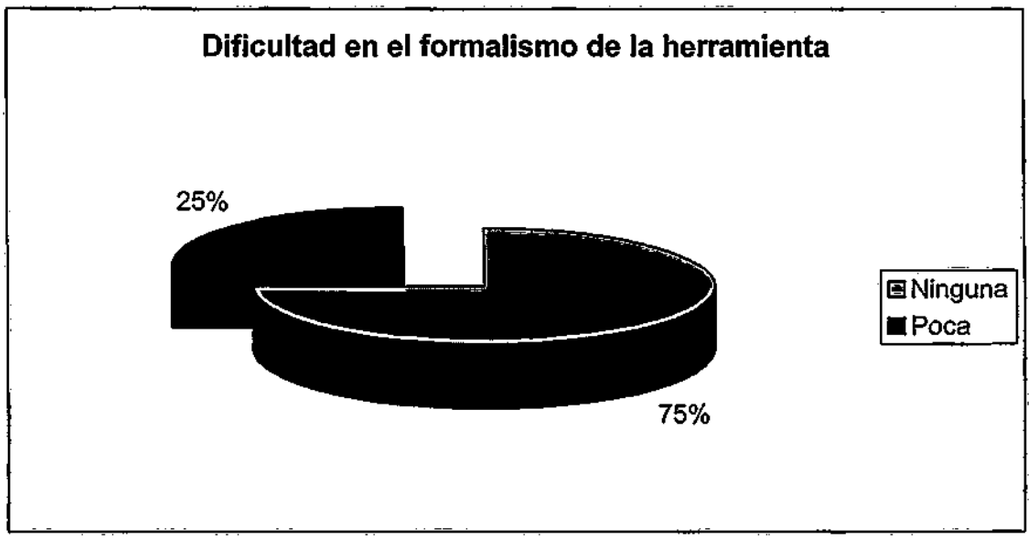


Figura 5.5: Dificultad observada por los usuarios en el formalismo utilizado por la herramienta

Con respecto a lo que les ha resultado más difícil de llevar a cabo a los usuarios, la mayoría de ellos no han argumentado nada en especial, habiendo completado en la totalidad de los casos estudiados la experiencia sin problemas mayores.

Por otro lado, un dato relevante a tener en cuenta es la utilidad que piensan los usuarios que una herramienta de este tipo pudiera tener para su labor cotidiana, además de pensar en algún dominio donde crean que la herramienta pudiera resultar útil. La gráfica que se representa en la Figura 5.6 muestra la opinión de los usuarios sobre la utilidad de la herramienta. Igualmente, a partir de los datos de la encuesta, se ha

observado como los usuarios apuntan a posibles dominios que han considerado útiles para la aplicación de la herramienta, como el dominio colaborativo de la docencia, el mantenimiento de material docente on-line, creación de laboratorios virtuales, y mantenimiento de páginas web institucionales.

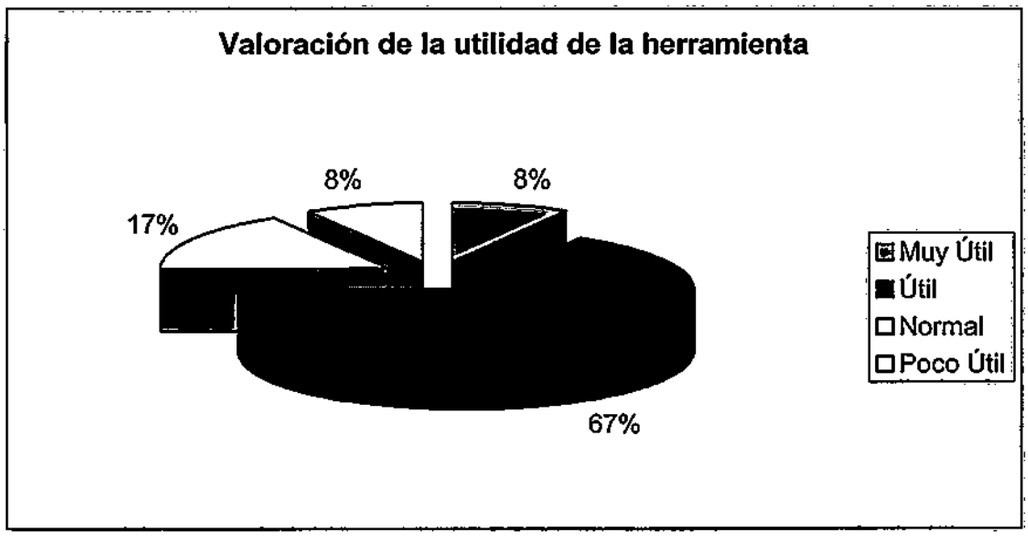


Figura 5.6: Valoración de la utilidad de la herramienta por parte de los usuarios

En cuanto a las carencias encontradas y relacionadas directamente con las acciones de edición, los usuarios no encontraron limitación sobre lo que podían hacer para llevar a cabo su fin que, como puede comprobarse por el número de primitivas generadas, difiere de un usuario a otro dependiendo del camino tomado para llegar a tal fin. Por otro lado, los usuarios tuvieron percepción de que lo que pretendía hacer coincidió con lo que hicieron realmente y, quitando algunas matizaciones que se darán más adelante, los usuarios no tuvieron noción de grandes carencias y por lo tanto no se vieron afectados en ninguna medida al respecto para llevar a cabo la experiencia.

5.2.3.3 Conclusiones finales sobre la experiencia

A partir la experiencia llevada a cabo con los usuarios, se desprenden conclusiones referentes tanto a la usabilidad como a aspectos cualitativos que afectan directamente al rendimiento de la propia herramienta en sí.

Por una parte, el hecho de que los usuarios estuvieran inicialmente adiestrados con herramientas que edición similares, ha permitido el centrar la atención especialmente en la capacidad del entorno de edición WYSIWYG de DESK. Los resultados a este respecto han sido prometedores, en el sentido en que los usuarios han utilizado poco tiempo para llevar a cabo la experiencia. Así mismo, la facilidad de uso de la herramienta, puesta de manifiesto por los usuarios, ha hecho que éstos se sintieran

cómodos y seguros en la edición, recordándoles el entorno de trabajo, a rasgos generales, a otras herramientas de edición estándar para la web de gran difusión y utilización habitual. En cualquier caso, y para aportar quizás una nota de realismo sin pérdida de la perspectiva y objetivos de la experiencia, hay que decir que estos aspectos pueden ser por otro lado bastante dependientes de la experiencia en concreto presentada a los usuarios, que aunque por otro lado se ha tratado de que esta sea lo más amplia posible y de que represente, de una manera sencilla, algunas de las posibilidades de DESK, es posible que cambiando el marco de la experiencia los resultados pudieran variar sensiblemente. No obstante el objetivo está cumplido con creces si podemos afirmar, como es el caso, que en media la herramienta resulta asequible a los usuarios y que además ésta resulta intuitiva y fácil de usar para un usuario final. De esta misma forma, otro dato representativo en el estudio demuestra que el formalismo empleado para operar bajo DESK también resulta fácil de seguir por parte de los usuarios, algo que sigue corroborando en cierta medida lo argumentado anteriormente.

Respecto a la información inferida por DESK para llevar a cabo los cambios efectuados por los usuarios, se ve cómo cada usuario ha seguido su propio camino para completar la experiencia, algo que se puede observar a partir del número de primitivas del modelo generadas. En este sentido, algunos usuarios más diestros aprovecharon las características ofrecidas por el entorno de edición para hacer más provechoso cada paso de la experiencia, siendo por tanto un entorno que permite, para cualquier número de acciones que lleve a cabo el usuario, llegar al mismo fin. De esta forma es posible inferir con éxito, como muestran las estadísticas, cada una de las primitivas, partiendo de la idea de que hay usuarios más diestros y otros que necesitan un número de pasos intermedio más elevado para conseguir el mismo resultado.

En cualquier caso, y a pesar del alto índice de acierto de la herramienta, es necesario decir que éste no fue el ideal, y que durante la experiencia los usuarios fueron conscientes de algunos errores de la herramienta en el manejo de ciertos objetos de la presentación, como tablas y listas y saltos de carro que la herramienta, en algunos casos, no supo inferir correctamente para resolver la ambigüedad inherente al proceso. La causa de estos fallos, en gran parte, fue la incapacidad de la herramienta para llevar a cabo cambios en las posiciones elegidas por el usuario a la hora de detectar o conservar esta posición en el modelo de la presentación, es decir, las acciones fueron identificadas y, en su mayoría, el contexto semántico fue localizado, sin embargo no se pudo determinar en qué lugar realizar inserciones de, por ejemplo, retornos de carro, inserciones de elementos, etc., debido a la dificultad de manipular información entre los bloques resultado de la renderización de relaciones multivaluadas del dominio o de objetos cuya localización de forma individual no fue posible. En cualquier caso, y aunque en el futuro se hará frente a este tipo de ambigüedad, el índice de acierto fue razonablemente alto y pocos fueron los usuarios que tuvieron noción de estos problemas. En cualquier caso, y como argumentación a lo anterior, los datos aportados por la encuesta reflejan que los usuarios no han tenido percepción de carencias en el

mecanismo seguido para llevar a cabo la experiencia, no habiéndoles afectado en medida alguna este hecho para lleva a cabo su fin, reportando una alta puntuación en cuanto a lo que la predicibilidad de la herramienta se refiere.

Concerniente a la utilidad que la herramienta les reporta a los propios usuarios, éstos han valorado positivamente la herramienta, aportando incluso distintos dominios en los cuales ven una clara aplicación de la herramienta como ayuda para su labor diaria. Así mismo, los usuarios han apuntado a la posibilidad de integrar la herramienta con otros estándares, consiguiendo así incrementar la potencia de uso y la comodidad frente a ciertas tareas cotidianas en distintos dominios de aplicación. Entre los dominios más valorados se encuentran el de la docencia, material didáctico y cursos on-line, así como el aspecto colaborativo de estos dominios, como la gestión de documentación en grupo, prácticas de laboratorio tutorizadas, laboratorios virtuales y páginas web institucionales. Esto pone de manifiesto una idea que será presentada más adelante como alternativa de trabajo futuro y que liga, como muchos usuarios han sugerido, con la aplicación en el campo colaborativo mediante un control de la sesión a partir de la creación de sesiones públicas y privadas de edición, poniendo un especial énfasis en el tratamiento de un modelo dinámico del usuario.

Respecto a las posibles carencias del entorno de edición encontradas por los usuarios, podemos destacar el escaso número de botones y asociaciones entre acciones típicas de un entorno comercial y completo de edición, la definición de ciertas teclas aceleradas, iconos en la barra de herramientas, el poder modificar ciertas propiedades de algunos objetos de la presentación una vez creados en el documento, así como hacer más intuitiva la manipulación de algunos widgets. En general, este tipo de sugerencias apuntan una vez más a la abstracción del usuario sobre la herramienta al intentar compararla de forma intuitiva con algunos de los editores-navegadores comerciales existentes en el mercado. Si bien es cierto que nunca fue el propósito de este trabajo el hacer un producto comercial, también es cierto que se atendió con mayor dedicación a la parte de inferencia y heurísticas, dejando quizás un poco de lado la parte de la interfaz si la comparamos con una aplicación comercial, pero incorporando sin embargo la suficiente funcionalidad como para llevar a cabo los cambios más comunes sobre documentos web. En cualquier caso, este tipo de cuestiones también serán tomadas en consideración para una futura ampliación de las características y la funcionalidad de la herramienta.

5.3 Discusión y comparativa con otros sistemas

El desarrollo de una herramienta de autor WYSIWYG para páginas dinámicas es un problema inherentemente difícil, puesto que de lo que se trata, al fin y al cabo, es de cambiar el mecanismo de generación de las propias páginas web, algo que en la mayoría

de los casos no puede ser llevado a término sin manipular directamente la información procedural relativa a la generación (i.e. lenguajes de programación para la web).

El trabajo presentado en esta tesis apuesta por una visión multi-paradigma, donde entran en juego distintos campos de investigación en los que ha habido notables aportaciones (mirar Capítulo II. Trabajo relacionado) y sobre los que es necesario hablar, poniendo de relieve las diferencias y aportaciones de esta tesis sobre dichos campos y herramientas. De entre los campos mencionados, las aportaciones más importantes de DESK se sitúan en los siguientes: La Programación por Demostración y los Sistemas de Extracción del Conocimiento.

DESK permite a un usuario final modificar una página web concreta generada por PEGASUS, cambiando la herramienta la forma en la que se generará en el futuro esa página. Para llevar esto a término, se monitoriza constantemente al usuario bajo un entorno de edición y navegación web, abstrayendo así información sobre sus acciones e infiriendo, posteriormente, cambios que afectan a los modelos subyacentes de PEGASUS, y por tanto a la forma en la que este sistema generarán las futuras páginas web. Esto es por tanto un problema relacionado con el paradigma de la Programación por Demostración [Cyp93], [CS99a], [ACMC00], donde el sistema infiere información procedural a partir de ejemplos de lo que el usuario quiere llevar a cabo.

Existen algunas aportaciones que hacen uso de las técnicas descritas en el párrafo anterior, como AVANTI [PSS01] que monitoriza la interacción del usuario bajo un entorno de navegación web orientado a la accesibilidad, adaptándose éste a las características del usuario durante la navegación. Otros, como WebSheets [WSC02] analiza las acciones del usuario para generar un lenguaje estructurado de consultas denominado QBE [Zlo77], utilizado para realizar accesos a bases de datos de forma automática, y evitando así que el usuario tenga que escribir directamente la *queries*.

Turquoise [MM97] y Scrapbook [SK98] son también ejemplos de navegadores inteligentes, donde se detecta cuáles son los fragmentos de páginas web en los que el usuario está interesado. El resultado de la interacción es la composición de una página, a modo de *collage*, donde cada uno de los fragmentos que la componen son actualizados automáticamente con información de la web.

La principal diferencia con estos sistemas es que DESK monitoriza al usuario durante la interacción en un entorno de edición completamente WYSIWYG, a diferencia de estas herramientas que son en su mayoría navegadores y no editores, como por ejemplo AVANTI. En el caso de Turquoise y Scrapbook no existe un interfaz como tal, sino que el usuario selecciona y copia fragmentos de páginas a una ventana y el sistema genera automáticamente los *scripts* de actualización. Por el contrario, WebSheets sí está basado en un entorno de edición WYSIWY, aunque esta herramienta está exclusivamente enfocada a la creación de tablas para la generación automática, por

demostración, de las *queries* en SQL. En cambio, en DESK un autor tiene libertad para cambiar y editar la distintos elementos de la página, así como insertar y borrar controles HTML (tablas, listas de selección, combo box), infiriendo el sistema automáticamente los cambios.

Otras de las características de DESK que lo acercan al paradigma de la Programación por Demostración, es la aportación de un agente de asistencia basado en la idea del Agente de Información [IAG] utilizado en el sistema TriIas [BDP00], cuya idea fue inicialmente esbozada en [LN99], donde el usuario interactúa con una aplicación web, y donde el agente planifica la navegación del usuario. Este sistema utiliza tecnología wrapper [HS00], [Mus99], [GRV+98], entrenándolos para buscar fragmentos relevantes en documentos web. Por el contrario, el agente de asistencia DESK está basado en mecanismos de inferencia muy simples, mediante el análisis y caracterización de la información disponible, y sin utilizar algoritmos de aprendizaje automático de comportamiento o modelos probabilísticos, como en el asistente de Microsoft Office [HBH+98], para el control de la inferencia. SMARTEdit [Lie01] es otro ejemplo de este tipo de aportaciones, donde utilizando técnicas de aprendizaje automático el sistema infiere información procedural durante el proceso de edición, a partir de la monitorización del usuario durante la interacción. De forma muy parecida, TELS [MW92] tiene en cuenta la interacción del usuario en un entorno de edición de texto, abstrayendo patrones de la iteración del usuario para la detección de bucles y condiciones, lo que le permite llevar a cabo tareas complejas. En esta herramienta el usuario puede intervenir para corregir la información inferida en el caso de que esta no sea correcta. En DESK el usuario se abstrae completamente de los mecanismos de inferencia utilizados, siendo responsabilidad del sistema resolver posibles ambigüedades utilizando conocimiento del dominio.

Un rango distintivo en DESK es la utilización de un modelo del dominio para la caracterización de cambios. En la literatura relativa a Programación por Demostración esta característica ha sido poco explotada en general [MC03b]. Peridot [Mye88] fue uno de los primeros sistemas en utilizar un modelo de datos, muy simple, que permite al usuario crear una lista de ejemplos para construir los widgets de la interfaz. En Gold [MGG94] y Sagebrush [Eet94], el usuario puede construir gráficos a medida relacionando elementos visuales y propiedades a conjuntos de datos. El modelo de datos de Peridot consiste en una lista de tipos de datos primitivos. Gold y Sagebrush utilizan un modelo relacional. En DESK se trata de disponer de estructuras más ricas en información semántica, utilizando la ontología y la red semántica de instancias del dominio de PEGASUS, lo que permite analizar relaciones entre los distintos objetos que componen la presentación para una caracterización y un tratamiento de la ambigüedad más eficaz. Otro antecedente es HandsOn [CS99b], donde el diseñador puede manipular ejemplos explícitos de datos de la aplicación en tiempo de diseño para construir visualizaciones dinámicas que dependen de los datos de la aplicación en tiempo de

ejecución. A ese respecto, DESK utiliza un modelo de conocimiento de la aplicación más expresivo que HandOn que, por otro lado, es completamente transparente al usuario a partir del principio WYSIWYG adoptado para la construcción de la herramienta, lo cual evita al usuario preocuparse de la representación interna del conocimiento utilizada.

La Programación por Demostración, conlleva una ambigüedad inherente al hecho de que se está derivando información general a partir de casos particulares proporcionados por un humano. Para resolver esta ambigüedad se han utilizado en otros sistemas [Lie99] estrategias como el seguimiento de las acciones del usuario paso a paso (vs. observar sólo el estado inicial y final), la utilización de varios ejemplos, ejemplos negativos, o la petición interactiva de ayuda al usuario, entre otras. Este es el caso de DESK, que utiliza un modelo de monitorización como método de seguimiento para obtener información enriquecida de las acciones del usuario en cada paso, pudiendo hacer frente a casos de ambigüedad que la herramienta sabe resolver automáticamente.

Inicialmente, DESK fue ideado bajo la posibilidad de realizar la inferencia de los cambios del usuario sin utilizar una herramienta de autor específica [MC02a]. Se propuso utilizar un editor web estándar para realizar la inferencia a partir de comparar la página antes y después de ser modificada, de forma que el sistema pueda averiguar los cambios automáticamente y localizarlos en los modelos subyacentes. Sin embargo esta propuesta no era demasiado viable, debido a la necesidad de utilizar complejos algoritmos de comparación de páginas HTML [DB96], las cuales pierden parte de su estructura cada vez que son editadas desde un editor web estándar. De esta forma, en DESK se propusieron inicialmente otras técnicas de Programación por Demostración, no basadas en la monitorización directa del usuario y el análisis del posible modelo de tareas de bajo nivel, sino a partir de un solo ejemplo de estados inicial y final (como en [MM99] y [FF94]), ayudándose de la misma forma del modelo del dominio para dotar de significado a los cambios detectados. Por otro lado, la utilización de una técnica de este tipo tiene el inconveniente de no existir un límite entre la cantidad de acciones que se pueden llevar a cabo entre el estado inicial y el final, con la posibilidad de perder parte de la información intermedia. En el caso de páginas web, este factor se veía adicionalmente perjudicado por la peculiaridad de eliminar y añadir código HTML que tienen algunos editores HTML convencionales. Finalmente se optó por desarrollar la propuesta actual narrada en esta tesis, donde se apostó por la monitorización del usuario mediante una herramienta de autor específica como medio de garantizar un mayor control sobre los cambios y una caracterización conceptual de los mismos más eficaz.

Para poder inferir información relativa a los cambios que el usuario lleva a cabo en el documento web generado por PEGASUS, DESK primeramente asocia las acciones del usuario, reflejadas en el modelo de monitorización, a información sintáctica de contexto extraída de la propia página web que el usuario edita. Esta

información será utilizada en el futuro para localizar objetos en el modelo del dominio, caracterizando los cambios que lleva a cabo el usuario con información semántica, para luego acometerlos y cambiar la forma en la que se generarán las futuras páginas web. De la misma forma, el agente de inferencia de DESK caracteriza, mediante información del dominio, controles HTML asociadas a relaciones y objetos, permitiendo así realizar cambios automáticos entre elementos de la presentación a partir de la detección de patrones de iteración del usuario en la edición del documento. El análisis de los documentos generados por PEGASUS es pues un elemento clave en los problemas abordados. En definitiva, se trata de un problema de Extracción Automática del Conocimiento de Páginas Web como el que llevan a cabo los wrappers.

Los wrappers proporcionan un mecanismo de acceso uniforme a la información almacenada en la web, localizada en repositorios heterogéneos, extrayéndola y dándole una estructura para un posterior procesamiento automático de la misma. En términos generales, las técnicas que los wrappers utilizan están basadas en el procesamiento del lenguaje natural y en el reconocimiento de ciertos elementos de las páginas HTML basados en el *layout* y el estilo de página. En DESK hay un elemento que en general no aparece en estos sistemas, se trata de un usuario actuando sobre el documento web. De esta forma, la extracción del conocimiento en DESK se lleva cabo a partir de las acciones del usuario sobre las partes del documento que éste modifica. Por otro lado los wrappers prescinden de la utilización de un modelo de datos, pues en la mayoría de los casos lo infieren ellos mismos o lo tienen ya pre-programado. En DESK se dispone de información explícita relativa al dominio y a la presentación, lo que facilita la identificación de los objetos origen constructores de la presentación. DESK no realiza un tratamiento del lenguaje natural, así el tratamiento del HTML se vuelve más sencillo al disponer de conocimiento del dominio, algo a lo que los wrappers no tienen acceso por normal general.

Los wrappers son utilizados ampliamente para localizar fragmentos concretos dentro de documentos o, incluso, ciertas estructuras, como en el caso de [CHJ02], donde se propone un mecanismo de aprendizaje para el reconocimiento de listas y tablas en documentos HTML, o trabajos como [BDP+00], [CTT00], [HurS00], donde se trata la problemática del reconocimiento de tablas en documentos web. A ese respecto, DESK lleva a cabo una modelización y tratamiento automático de los controles HTML incluidos en el documento web que el usuario edita. Sin embargo, a diferencia de los trabajos anteriores DESK no necesita reconocer la existencia de una tabla en sí, puesto que la tabla se supone creada e identificada (con un ID único), aunque al igual que en los trabajos anteriores, una vez detectada la tabla se procede a crear un modelo estructurado a partir de la geometría del widget, junto con otras características intrínsecas del mismo. Frente a estos sistemas basados en aprendizaje automático y modelos probabilísticos, DESK utiliza patrones de iteración muy simples, debido a que la herramienta dispone de información del dominio suficiente para poder identificar el

origen de los datos, por lo tanto no son necesarios lenguajes de extracción de información en el documento, ya que el agente investiga y extrae los datos que le interesan del modelo estructurado de monitorización.

5.4 Aportaciones

En resumen, las principales aportaciones de DESK, en función de cada uno de los campos de investigación puestos de manifiesto, tanto en este apartado como en el Capítulo II de esta tesis, son las siguientes:

- En el campo de los sistemas web hipermedia dinámicos: Una interfaz WYSIWYG que permite editar páginas web generadas dinámicamente por este tipo de sistemas (en concreto por PEGASUS), evitando al usuario final el tener que manipular directamente los lenguajes de especificación utilizados para la generación dinámica de documentos web. Además, gracias a los mecanismos de inferencia de la herramienta, es posible modificar tanto los contenidos como la presentación de una página web, requiriendo un mínimo esfuerzo por parte del autor.

- En el campo de la extracción automática del conocimiento: Un mecanismo de caracterización de objetos y partes integrantes de la presentación, mediante el uso de conocimiento del dominio. En este caso se hace relevante la búsqueda de información sintáctica de contexto dentro de la propia página HTML, así como la localización de semántica en el modelo de datos para la caracterización de cambios. Además, mediante un agente especializado, se analiza el modelo de acciones del usuario para extraer información sobre tablas, listas y otros elementos de la presentación, que serán modelizados atendiendo a sus características, pudiéndose realizar transformaciones automáticas entre estos elementos.

- En el campo de las interfaces de usuario: Una interfaz de usuario WYSIWYG inteligente basada en modelos, los cuales aprovecha para hacer inferencia y ayudar al usuario en la edición. El proceso de ingeniería inversa implícito que lleva a cabo la herramienta, aporta nuevos mecanismos para la utilización de información de los distintos modelos en el proceso de inferencia. Además, la modelización de las acciones del usuario, junto con información sintáctica y semántica, permite abrir nuevas puertas a la adquisición relevante de información del usuario final durante la interacción con la interfaz, permitiendo abstraer tareas de más alto nivel y consiguiendo aportar mayor inteligencia a este tipo de interfaces.

- En el campo de la Programación por Demostración: Un entorno WYSIWYG demostracional basado en la monitorización del usuario para minimizar la

ambigüedad y, a partir de un agente de inferencia, poder analizar patrones de iteración. Todo esto se lleva a cabo bajo el entorno de la web y utilizando conocimiento del dominio, lo cual permite dar cabida a nuevas posibilidades en la adquisición de conocimiento en entornos desmostracionales enfocados a la web. Otras técnicas ampliamente utilizadas en este paradigma, como la monitorización del usuario, se ven notablemente mejoradas a partir de la incorporación de información sintáctica y semántica sobe lo que el usuario edita, siendo el mecanismo de inferencia más eficiente y pudiendo tratar casos reales de ambigüedad de forma automática.

5.5 Recapitulación

A continuación se presentan la recapitulación y la discusión final de las principales aportaciones de esta tesis, materializadas en el trabajo descrito en el Capítulo III: Generación de documentos web dinámicos, y el Capítulo IV: Autoría mediante ejemplos.

La autoría de páginas web dinámicas descrita en esta tesis se basa en la utilidad de creación de herramientas de autor inteligentes que permitan flexibilidad en el diseño, haciendo especial hincapié en que no es fácil para un diseñador ajeno a una aplicación web el editar el diseño o los contenidos de la misma. Para hacer frente a este problema se ha mostrado un mecanismo englobado dentro de una herramienta de autor inteligente, DESK.

Seguidamente se detallarán unas conclusiones sobre DESK y las herramientas complementarias comentadas en esta tesis.

5.5.1 Autoría de documentos web dinámicos

La autoría de páginas web dinámicas se hace posible a partir de la creación de técnicas englobadas dentro de una herramienta de autor inteligente creada para ese propósito. DESK utiliza técnicas de Programación por Demostración para facilitar la interacción hombre-máquina, en tanto que el usuario sólamente tiene que proporcionar ejemplos de lo que quiere hacer en una página web para que la herramienta infiera finalmente los cambios en los mecanismos de generación.

La forma de trabajar de DESK se basa en la monitorización del usuario bajo el entorno de edición, construyendo para ello un modelo estructurado de monitorización, a partir de primitivas, que se utilizará posteriormente para inferir información más compleja, como cambios atómicos entre elementos de la presentación o, ya en el servidor, la modificación de estructuras del dominio y de la presentación de PEGASUS. Para elaborar el modelo de monitorización se utilizan heurísticas encargadas de localizar el contexto sintáctico (heurísticas de bajo nivel) y semántico (heurísticas de

alto nivel) de las acciones del usuario. Las heurísticas proporcionan, con la información disponible en cada una de las localizaciones (cliente o servidor), una forma de abstraer lo que el usuario intenta hacer al editar un documento HTML, automatizando el proceso y elaborando conocimiento de alto nivel a partir de la detección de ciertas estructuras especiales en la interfaz y meta-información procedente de PEGASUS. Esto permitirá inferir cambios con un mayor nivel de abstracción que los que involucran el cambiar ciertos atributos del código HTML, pudiendo incluso realizar cambios sobre reglas de presentación cuya aplicación recursiva las hacen difíciles de tratar directamente. El lenguaje de representación de las primitivas constructoras, y del modelo de monitorización en general, permite una alta escalabilidad, pudiendo añadir nuevas primitivas o incrementar la información semántica de las mismas en un momento dado. De esta forma, la estructura del modelo de monitorización aporta semántica sobre la interacción del usuario bajo la herramienta, pudiéndose usar posteriormente para la gestión de históricos o abstracción de modelos de tareas más complejos.

DESK proporciona ayuda al usuario mediante un agente de asistencia en la edición, el cual se encarga de detectar patrones de iteración mediante el análisis detallado del modelo de monitorización para así completar acciones que el usuario inició, como transformar un widget en otro, ordenar listas de elementos, etc. Para detectar estos patrones de iteración, el agente hace uso de algoritmos elaborados para la construcción de iteradores que permiten completar secuencias lineales, o en su defecto suministrar un pool de patrones de búsqueda para casos en los que no sea posible predecir un modelo de inducción. La configuración del agente es escalable, permitiendo definir nuevos patrones en el pool y cambiar las variables internas, siendo independiente en todo momento del dominio de la presentación.

DESK realiza el camino inverso seguido por PEGASUS para la generación de documentos HTML, partiendo del documento final e intentando obtener los objetos del dominio con que se construyó dicho documento. Esto supone un cierto grado de ambigüedad inherente que la herramienta trata de solventar mediante heurísticas de desambiguación, consistentes en la extracción de información de los modelos subyacentes. La ambigüedad producida puede surgir a distintos niveles, tomando DESK una solución predefinida de antemano para resolverla en todos los casos, incluida la intervención del usuario si finalmente no es posible deshacer la ambigüedad de forma automática. Por ejemplo, si se cambia el estilo de un elemento de una relación multivaluada (lista de elementos), DESK actúa cambiando el estilo de todos los elementos de la lista. Una solución más apropiada para este tipo de ambigüedad podría ser un mecanismo parecido al que se sigue con los patrones de iteración, donde se espera más de una intervención del usuario (valor de repetición en la configuración del agente) hasta que el sistema analiza todos los cambios antes de llevar a cabo el proceso de inferencia. Siguiendo esta estrategia, la ambigüedad en la toma de decisiones se reduce considerablemente al contar la herramienta con más información de alto nivel a

la hora de tomar decisiones de este tipo. Por otro lado, otro tipo de ambigüedades pueden ser resueltas a partir del cálculo del camino mínimo dentro de la red semántica de objetos del dominio (i.e. el camino más corto al objeto principal), así se puede caracterizar para un mismo objeto, por ejemplo "Velázquez", una expresión de acceso del tipo "El pintor barroco", frente a otras como "El autor de las Meninas en la Corte del Rey Felipe IV".

DESK parte del principio WYSIWYG que garantiza al usuario abstraerse de la representación interna del sistema en la generación dinámica de páginas web. Sin embargo, esto acarrea importantes limitaciones expresivas en el sentido de que no es posible crear nuevas unidades de conocimiento en la presentación. Esto significa, en particular, que puede ser problemático si el autor pretende realizar una edición completa, partiendo desde cero, de la totalidad del documento web, siendo imposible reconstruir desde ese punto de inicio los objetos y las relaciones del domino de forma automática. La razón de esta premisa es para ver cómo de lejos puede llegarse sin abandonar la filosofía WYSIWYG, aunque no hay nada que impida a DESK utilizar nuevas vistas del modelo de datos para aumentar su expresividad sacrificando, por otro lado, la facilidad de uso de la que se partió como requisito fundamental. En cualquier caso, el propósito de este trabajo no es diseñar una herramienta infalible, sino aportar una metodología para de la edición de aplicaciones web que cubra un amplio espectro de casos comunes.

En general, es imposible desarrollar un sistema de estas características que acierte siempre, a menos que se impongan tantas restricciones que el sistema pierda toda su generalidad, o se hagan tantas preguntas al usuario que deje de ser fácil de utilizar, por lo que la meta de esta filosofía consiste generalmente en alcanzar un grado de acierto suficiente para que el sistema basado en ejemplos resulte útil. Partiendo de esta idea, DESK puede inferir de forma satisfactoria los siguientes cambios en documentos web:

- Inserción, eliminación y modificación de fragmentos HTML, tanto en la plantilla de la presentación como en las reglas de generación.
- Inserción, eliminación y modificación de etiquetas HTML alrededor de elementos del dominio y reglas de generación.
- Creación, eliminación, modificación y transformación automática de cambios entre widgets de presentación como listas de selección, combo boxes y tablas.
- Inserción, eliminación, modificación y traslación de referencias textuales y multimedia del modelo del dominio.

Sin embargo hay otro tipo de cambios que DESK, por el momento, no puede llevar a cabo, como es la modificación de la estructura del dominio (relaciones entre las

unidades de conocimiento). Para ello un diseñador puede utilizar la herramienta de autor PERSEUS. Por el contrario, DESK permiten incluir de forma automática nuevas instancias del dominio, creadas de antemano, como objetos en la plantilla de la presentación.

Los tipos de cambio narrados anteriormente permiten ser llevados a cabo por un usuario poco experto de una forma sencilla e intuitiva, debido sin duda a la naturaleza WYSIWYG del entorno de edición. Sin embargo, existen otros cambios que pueden ser llevados a cabo por diseñadores más avanzados, con autorización explícita, los cuales involucran la gestión de perfiles de usuario a partir de la integración con HADES. De esta forma es posible adaptar el contenido de la presentación a un perfil de usuario, y además editarlo mediante DESK.

Existe un prototipo de la herramienta DESK, donde se ha implementado lo esencial de lo narrado en esta tesis. La herramienta de autor se ha probado sobre distintos escenarios de presentación de información, entre los que se incluyen dominios como catálogos para el comercio electrónico, información turística y de viajes, así como material educativo.

DESK ha sido desarrollado íntegramente sobre Java (JDK 1.3), utilizando adicionalmente otras tecnologías como XML/DOM.

5.5.2 Herramientas complementarias

PEGASUS, como sistema de representación del conocimiento propuesto, permite la especificación de la presentación separadamente de la construcción de contenidos, favoreciendo la reutilización y consistencia de la presentación, y reduciendo por tanto el coste de desarrollo. PEGASUS permite reproducir los modelos del dominio utilizados en una amplia gama de sistemas hipermedia. La generación dinámica de la presentación independientemente de la actualización del estado de la aplicación hace que PEGASUS sea compatible con herramientas de soporte para presentaciones adaptativas como las descritas en el capítulo de trabajo relacionado de esta tesis, pudiendo ser utilizado como un módulo complementario de estas herramientas. La generalidad se consigue proporcionando un marco de aplicación para la definición de ontologías que mejor se ajustan a un dominio o un autor concreto. La presentación de las páginas a generar se describe en términos de clases y relaciones de la ontología. Un modelo explícito de la presentación, separado de los contenidos, se utiliza para permitir a los diseñadores un extenso control sobre la generación de todos los aspectos de la presentación, con un coste de desarrollo moderado. Así mismo, PEGASUS permite incorporar elementos activos en sus plantillas de presentación, como reglas de presentación, que permiten modificar de forma global aspectos no triviales de la renderización de objetos del dominio, así como condiciones establecidas sobre modelos del usuario y/o de la plataforma.

La separación explícita entre contenidos y presentación llevada a cabo por PEGASUS, hace posible que herramientas como DESK puedan intervenir en la edición de estos dos modelos por separado, a partir de la modificación de una página web generada dinámicamente.

Si bien la construcción de plantillas está al alcance de cualquier diseñador de páginas web familiarizado con las tecnologías HTML y JSP, la definición de ontologías es una tarea delicada que requiere la participación de un diseñador avanzado, entrenado en el uso de la herramienta. Sin embargo el desarrollo de herramientas de autor como PERSEUS o DESK hacen que esta fase de diseño pueda ser llevada a cabo por autores no expertos en ningún lenguaje de especificación en especial. De esta forma PERSEUS permite trabajar en conjunción con PEGASUS para alejar al diseñador de utilizar un lenguaje de programación específico. Mediante el entorno de edición de PERSEUS un diseñador puede disponer la información de la ontología en ventanas separadas, pudiendo navegar libremente por la jerarquía que forma la ontología. Posteriormente, y siguiendo la misma filosofía, el autor puede proceder a crear instancias y, por tanto, una red semántica de objetos del dominio de una forma fácil, aplicando el paradigma de la reusabilidad en todo momento. PERSEUS se encarga de generar automáticamente un modelo del dominio compatible con el lenguaje de modelado de PEGASUS, centrando al diseñador en la tarea de creación y vinculación de fragmentos multimedia y de composición del conocimiento de una interfaz web. Aunque PEGASUS y PERSEUS utiliza un pseudo lenguaje basado en XML, parecido al RDF, para la construcción del modelo dominio, éste no está basado en ningún estándar en concreto dentro de los lenguajes establecidos para la web semántica [W3C], lo cual puede afectar a la transportabilidad de la ontología del dominio para la compartición de conocimiento con otros sistemas similares.

Por otro lado, el portal HADES se encarga de la gestión de módulos entre las distintas herramientas de creación y autoría de presentaciones dinámicas, controlando el flujo y aportando un control de la sesión en la interacción con el usuario, para canalizar de forma homogénea el acceso a las distintas páginas web de autor, las cuales pueden ser creadas por distintos diseñadores y ser accedidas por distintos usuarios en cada momento.

Existen prototipos de PEGASUS, PERSEUS y HADES, donde se ha implementado gran parte de lo narrado en esta tesis. Estas herramientas ha sido probadas sobre distintos escenarios de presentaciones, entre los que se incluyen dominios como catálogos para el comercio electrónico, información turística y de viajes, así como material educativo.

Tanto PEGASUS, PERSEUS como HADES han sido desarrollados íntegramente sobre Java (JDK 1.3), utilizando adicionalmente otras tecnologías como XML/DOM y JavaServer Pages.

5.6 Trabajo Futuro

A partir del trabajo descrito en la memoria de esta tesis, se proponen una serie de ampliaciones que pueden verse a su vez como mejoras sobre el trabajo presentado, estando algunas de estas ampliaciones dirigidas a suplir las limitaciones presentadas y comentadas en anteriores apartados de este capítulo. Algunas de estas ampliaciones y mejoras pueden encauzar nuevas líneas de investigación a seguir en el área de aplicación del trabajo. Sin ningún orden en concreto, las propuestas de trabajo futuro son las que se describen a continuación:

- Mejora del modelo del dominio con la aportación de algún lenguaje enfocado a la web semántica. El llevar a término esta aportación podría afectar positivamente a la capacidad expresiva en sí del lenguaje de descripción del dominio. Este lenguaje, tal y como se indicó en capítulos anteriores, está basado en una especificación destinada a la definición de ontologías y objetos del dominio. El lenguaje actual utilizado es un lenguaje ad-hoc similar a RDF [RDF], aunque sin alcanzar la potencia semántica y expresividad de éste. La propuesta consiste en permitir a PERSEUS generar, a partir del diseño llevado a cabo por el usuario, el lenguaje de la web semántica elegido, así como permitir a PEGASUS y DESK el poder trabajar con un modelo del dominio expresado íntegramente en algún lenguaje estándar de la web semántica, como RDF o RDFS, u otros como DAML+OIL [DO01], aprovechando así la adecuación de estos lenguajes para el propósito del trabajo presentado. Otra ventaja al respecto de especificar el conocimiento en estos lenguajes de la web es su naturaleza estándar, lo que permitirá la transportabilidad del propio modelo del dominio para su integración en otros sistemas y herramientas de edición estándares [W3C], con la idea fundamental de compartir el conocimiento disponible y escalarlo en la medida de lo posible.

- Mejora del Modelo de la Presentación. Bajo esta propuesta se enmarca la idea de poder construir elementos de la presentación con mayor potencia funcional, a partir de la definición de controles HTML más complejos, como por ejemplo árboles colapsables y expandibles, controles dinámicos embebidos en otros lenguajes de programación, etc. Desde ese punto de vista, PEGASUS permitiría una mejor renderización de los objetos del dominio y DESK, por otra parte, permitiría inferir los cambios sobre dichos objetos adecuándolos a la incorporación de nuevos componentes de la interfaz acordes con la tecnología actual. Es posible que, finalmente, para aumentar la capacidad expresiva del lenguaje de presentación sin aumentar también la ambigüedad en la inferencia, sea necesario crear un modelo explícito de la presentación, como se propone en [VBS01]. Crear un modelo con mayor abstracción permitirá llevar a cabo la especificación de componentes más complejos, abstrayéndose el sistema del lenguaje de especificación utilizado para la representación actual del modelo (i.e. JSP [JSP]), lo que permitirá llevar a cabo la re-ingeniería inversa del proceso de generación de páginas web dinámicas de forma más

eficaz, reduciendo aún más la ambigüedad comentada en apartados anteriores, así como proporcionando mecanismos de integración con otros lenguajes de especificación existentes [PE02] para la creación de interfaces. A este respecto también sería interesante el poder construir una ontología de la presentación, donde se pudieran definir relaciones entre las distintas plantillas de la presentación, aumentando de esta forma la propia expresividad del modelo.

- Mejora del mecanismo de adición de nuevo conocimiento a los modelos del dominio y de la presentación. Esta propuesta surge a partir de una de las limitaciones de DESK comentadas anteriormente, basada en el hecho de que es imposible crear nuevas unidades de conocimiento u objetos de la presentación, así como construir una página partiendo desde cero. Disponiendo del modelo del dominio en tiempo de edición, se podrían evaluar las relaciones entre lo modificado y el contexto, buscando relaciones implícitas y creando nuevas unidades de conocimiento a partir del análisis de los objetos involucrado en el contexto más cercano al cambio realizado por el usuario. El sistema podría mantener un esqueleto de la meta-información del modelo para ir construyendo una ontología de cambios, parecida a la idea utilizada en la construcción de índices de estructuras con crecimiento dinámico. Esta misma información se podría utilizar en el caso de tener que generar nuevos objetos en la presentación, algo que sería fácil de hacer si se trata de crear objetos iguales a los existentes, o utilizando el esqueleto de cambios en el dominio que se genera durante el proceso y que el sistema mantiene en memoria para cada sesión.

- Toma en cuenta del modelo del usuario y de la plataforma de acceso. La idea principal en este caso es utilizar un modelo de usuario dinámico para el control de acceso múltiple por parte de varios usuarios a PEGASUS, apoyado en modelos existentes y cuya eficiencia ha sido probada previamente [Enc97a], [Enc97b]. De esta forma se controlaría la evolución del usuario durante la interacción, y DESK podría realizar ciertos cambios dependiendo del usuario presente en la edición, permitiendo incluso realizar simulaciones a partir de distintos usuarios que pudieran conectarse al sistema. Esto último podría llevarse a cabo teniendo acceso directo al modelo del usuario y también al de la plataforma de acceso, aunque habría que prescindir para estos casos de una edición puramente WYSIWYG para subir a un nivel de abstracción un tanto mayor, sin el cual sería imposible representar este tipo de acciones bajo un entorno de edición de estas características. Otra idea interesante, resultado de este planteamiento, es la utilización de distintas vistas personalizadas en la edición de una interfaz web, almacenando las modificaciones a partir del perfil del usuario y viendo, en cada caso, los cambios en la presentación a partir de dicho perfil. Esto permitiría un acceso y modificación colaborativa de las presentaciones, algo que se ya se ha esbozado en otros sistemas, como el presentado en [FCM99], donde distintos usuarios tienen acceso a acciones de edición persistentes de un documentos web, distinguiendo el sistema y presentando los cambios posteriormente

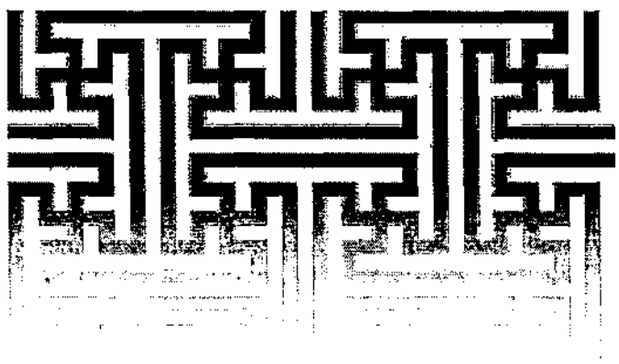
de forma distinta a cada usuario en función del su perfil. Sería interesante, en este mismo contexto, el poder también bloquear la edición sobre la plantilla de la presentación, permitiendo realizar ciertos cambios a partir del perfil del usuario y en base a distintos tipos de granularidad. Esto podría reducir la ambigüedad sensiblemente en el proceso de edición, al obligar a ciertos usuarios (quizás poco expertos) a realizar modificaciones sólamente sobre una granularidad más gruesa (p.e. directamente sobre las etiquetas externas en la plantilla que rodean a lo objetos de la presentación, sin alterar en lo más mínimo su funcionalidad interna).

- Creación de distintos editores de manipulación directa. Estas herramientas pueden estar integradas en el sistema (mediante HADES), o tal vez directamente en DESK, permitiendo editar aspectos como por ejemplo la composición de reglas de presentación, la configuración del agente de inferencia, o incluso la edición de ontologías utilizando una metodología más intuitiva (como en WebOnto [Dom98] o Protégé [NSD+01]). La filosofía de estas herramientas sería la de poder abstraer aspectos relativos a los objetos que se manipulan en cada caso, y proporcionar un entorno de manipulación directa para la creación y modificación de dichos objetos en la línea de trabajos anteriores como ATLAS [MC00], [MC01d]. Una de las aplicaciones más interesantes a este respecto sería la creación de patrones de iteración predefinidos por el usuario a partir de técnicas de Programación por Demostración, donde el usuario copia elementos entre widgets predefinidos y disponibles en la presentación y la herramienta genera directamente el patrón dentro del fichero de configuración del agente de inferencia. La idea principal de estas herramientas es que permitan la edición abstracta basada en modelos, si bien es verdad que esta ampliación disminuiría sensiblemente la característica WYSIWYG actual utilizada en el entorno DESK. Otra posibilidad al respecto consistiría en tener en DESK distintos tipos de vistas, separados quizás por una solapa o distintas ventanas, donde se dispusiera de una visión WYSIWYG de la edición y otro tipo de vistas con un nivel de abstracción mayor para la edición directa de las plantillas de la presentación o incluso del modelo de la presentación, pudiendo disponer de ciertos modelos subyacentes en tiempo de diseño y permitiendo, bajo un control del modelo del usuario, que ciertos usuarios puedan ver la información final generada como si de una plantilla se tratara, señalando claramente la separación entre lo referente a presentación y los propios objetos renderizados en la presentación.

- Ampliar la capacidad del agente de inferencia. La ampliación de la capacidad del agente permitirá reconocer nuevas transformaciones entre widgets o, incluso, abstraer un modelo de alto nivel de tareas del usuario para poder inferir mayor conocimiento sobre el comportamiento del usuario en la aplicación. Otros trabajos han basado su investigación en este área (ver [GCR+98], [GM99]), los cuales pueden ser de ayuda para la construcción de agentes personalizados o que, mediante técnicas más avanzadas de wrappers o Semantic Web Mining, permitan encontrar conocimiento

sobre distintas acciones del usuario en varios documentos web modificados sobre la misma presentación o incluso sobre presentaciones anteriores, disponiendo así de un histórico de cambios cuya información puede utilizarse para generalizar ciertas acciones a partir de la detección de distintos patrones de iteración más complejos sobre la edición [Lie99]. Para ello la funcionalidad del agente podría mejorarse en función de otros trabajos relacionados más orientados al paradigma de los Agentes de Información [BDP+00] o Agentes de Reconocimiento [LFW01].

- Posibilidad de realizar la inferencia de los cambios del usuario sin utilizar una herramienta de autor específica. Se trataría de retomar la idea inicial bajo la cual fue concebida DESK, es decir, utilizar un editor web estándar en vez de una herramienta específica para la autoría de página. Para ello DESK front-end dejaría de ser una interfaz de edición para convertirse en un *plug-in* conectado a un navegador convencional. De esta forma la inferencia se llevaría a término en el servidor a partir de la página enviada por el conector en el cliente, analizando los estados inicial y final de la página web, es decir, comparando la página antes y después de ser modificada de forma que el sistema pueda averiguar y acometer los cambios automáticamente. Esto, como ya se comentó, tiene problemas debido a la complejidad de comparar dos páginas HTML (ver Apartado 5.3). En cualquier caso, y si la tecnología lo permite en poco tiempo, lo mejor sería poder utilizar esta propuesta, en vez de monitorizar quizás al usuario constantemente en una herramienta de edición creada a propósito, dejando al usuario libertad para utilizar las herramientas de navegación y edición que desee.



ANEXO

En este anexo se incluye información adicional sobre las pruebas llevadas a cabo con los usuario: 1) Cuestionario presentado al usuario para la evaluación de la experiencia con DESK, 2) Página inicial presentada al usuario para la experiencia con DESK y 3) Página final presentada al usuario para la experiencia con DESK.

1. Cuestionario presentado para la evaluación de la experiencia de autoría con DESK

1) ¿Cuánta dificultad le ha supuesto hacerse con el formalismo durante el adiestramiento llevado a cabo para la utilización de la herramienta?

☐ Ninguna
☐ Poca
☐ Media
☐ Mucha
☐ Máxima

2) ¿Estaba previamente adiestrado con herramientas similares?. En caso afirmativo especifique cuales.

3) ¿Qué es lo que le ha resultado más difícil de llevar a cabo de la experiencia? ¿Por qué?

4) ¿Echó en falta algo durante la experiencia? ¿Qué piensa que podría aportar más usabilidad a la herramienta para llevar a cabo su cometido?

5) ¿Cómo valoraría de 0 a 5 la predecibilidad de la herramienta?

6) ¿En los casos en que ha tenido percepción de fallo por parte de la herramienta, ha estado relacionado con alguno de estos aspectos? (Indique el nº de fallos ocurridos)

☐ Tratamiento de bloques y fragmentos HTML
☐ Cambios de estilo
☐ Manejo de tablas
☐ Manejo de listas de elementos
☐ Imposibilidad de añadir nuevos contenidos
☐ Otros. Especificar:

7) En relación con las preguntas anteriores, ¿Cree que lo que pretendía hacer coincide con las posibilidades que le ha ofrecido la herramienta durante la edición?, ¿Encuentra alguna carencia al respecto?, ¿En qué medida le ha afectado?.

8) En general, una vez aprendido el uso de la herramienta, ¿cómo valora la facilidad de manejo de la herramienta?

☐ Muy fácil de utilizar
☐ Fácil de utilizar
☐ Dificultad normal
☐ Difícil de utilizar
☐ Imposible de utilizar

9) ¿Piensa que la herramienta puede ser útil bajo algún dominio que usted imagine?, ¿Cuál?

10) ¿Cree que la herramienta utilizada le puede resultar útil en su actividad cotidiana?

☐ Si, muy útil

☐ Si, útil

☐ De utilidad normal

☐ Poco útil

☐ Completamente inútil

11) Otros comentarios o sugerencias

2. Página web inicial presentada al usuario para la experiencia de autoría con DESK

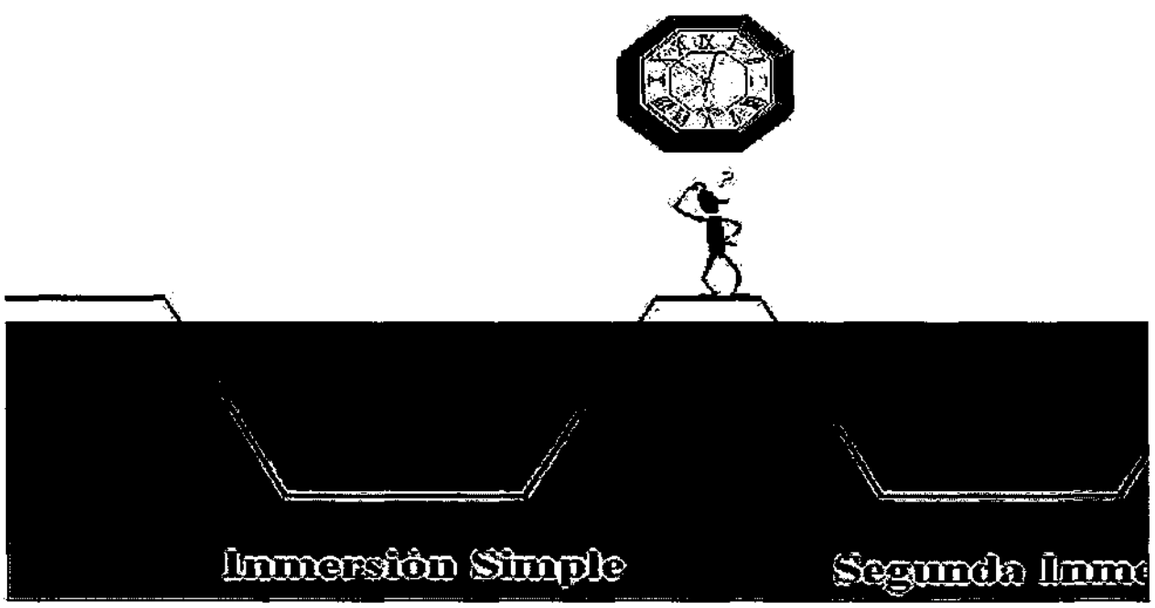
Conceptos Previos

- Nuestro Primer Contacto con el Agua
- Los Efectos de los Cambios de Presion
  - Nuestro Primer Contacto con el Agua
- La Respiracion con la Escafandra Autonoma
  - Nuestro Primer Contacto con el Agua
  - Los Efectos de los Cambios de Presion
    - Nuestro Primer Contacto con el Agua
- Las Liminaciones de Respirar Aire a mas de una Atmosfera de Presion
  - Nuestro Primer Contacto con el Agua
  - Los Efectos de los Cambios de Presion
    - Nuestro Primer Contacto con el Agua
  - La Respiracion con la Escafandra Autonoma
    - Nuestro Primer Contacto con el Agua
    - Los Efectos de los Cambios de Presion
      - Nuestro Primer Contacto con el Agua

Planificacion, Control y Estudio de las Inmersiones

Concepto

En este capitulo conoceremos como se puede bucear alejado del riesgo de la sobresaturacion critica de nitrogeno, lo que llamamos bucear sin rebasar la curva de seguridad.




Más de 12 Horas => La segunda es simple


Menos de 12 horas y más de 10 minutos => La segunda es sucesiva


Menos de 10 minutos => La segunda es continuada

OBJETIVOS
i. *Conocer que es la curva de seguridad.* ii. *Conocer a que llamamos tiempo en el fondo* iii. *Conocer a que llamamos tiempo limite* iv. *Conocer a que llamamos profundidad de la immersion* v. *Conocer a que llamamos frontera de seguridad* vi. *Conocer cuales son los factores que incrementan el reisgo de la E.D.*

Puntos a tener en cuenta

Inmersiones Simples

Explicacion
Son aquellas en las que la ultima inmersion se llevo a cabo en mas de 12 horas.
Actuaremos normalmente, como si se tratara de una primera inmersion

Inmersiones Continuadas

Explicacion
Son aquellas que se llevan a cabo en menos de 10 minutos desde la ultima inmersion.
En este tipo de inmersiones, la inmersion que se ha considerado como segunda en la Figura de arriba se convertira automaticamente en la continuacion de la primera.

Inmersiones Sucesivas

Explicacion
Aquellas en las que la ultima inmersion se llevo a cabo en mas de 10 minutos pero menos de 12 horas de la anterior

El tiempo limite de una inmersion sucesiva es **tl = TL-TNR**
Para calcular este tiempo limite (tl) tendremos que calcular el tiempo limite de una inmersion sencilla (TL) a esa profundidad
(Mediante la Tabla del Tiempo Limite y Coeficientes de Salida) y restarle el tiempo de nitrogeno residual (TNR)

Tablas de Inmersion

Tiempo Limite y Coeficiente de Salida
Coeficientes de Nitrogeno Residual

Macias, as Designer you can modify this presentation, please insert the XML file:

XML Monitoring Model:

Send Changes

3. Página web final presentada al usuario para la experiencia de autoría con DESK

Conceptos Previos

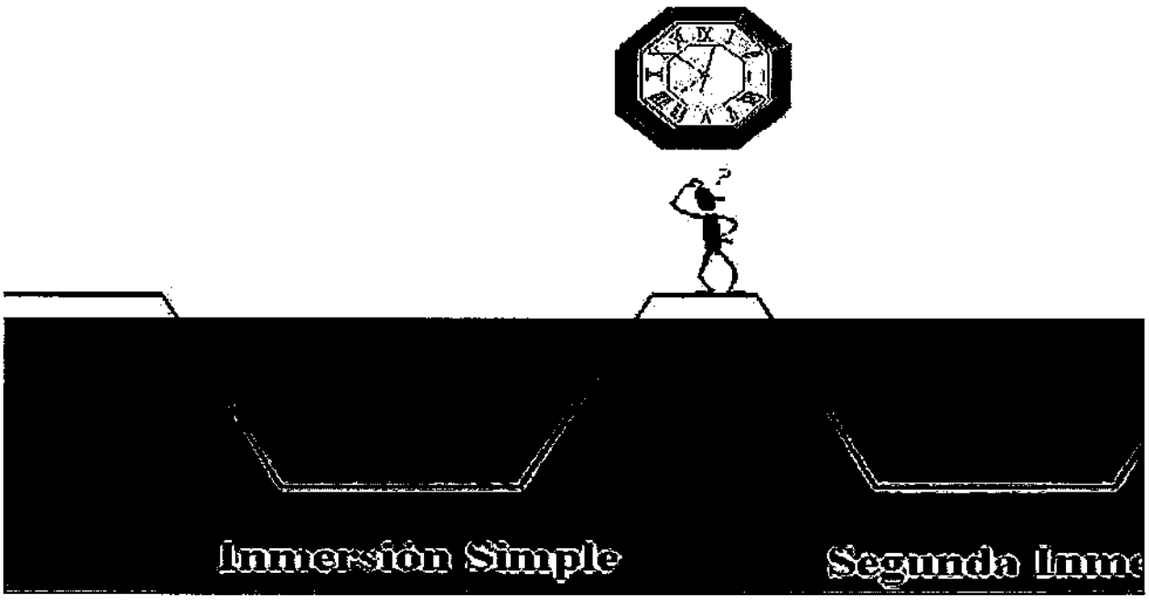
- Nuestro Primer Contacto con el Agua
- Los Efectos de los Cambios de Presion
  - Nuestro Primer Contacto con el Agua
- La Respiracion con la Escafandra Autonoma
  - Nuestro Primer Contacto con el Agua
  - Los Efectos de los Cambios de Presion
    - Nuestro Primer Contacto con el Agua
- Las Liminaciones de Respirar Aire a mas de una Atmosfera de Presion
  - Nuestro Primer Contacto con el Agua
  - Los Efectos de los Cambios de Presion
    - Nuestro Primer Contacto con el Agua
  - La Respiracion con la Escafandra Autonoma
    - Nuestro Primer Contacto con el Agua
    - Los Efectos de los Cambios de Presion
      - Nuestro Primer Contacto con el Agua

Planificacion y Control de las Inmersiones

Concepto

En este capitulo conoceremos como se puede bucear alejado del riesgo de la sobresaturacion critica de nitrogeno, lo que llamamos bucear sin rebasar la curva de seguridad.

Conocer que es la curva de seguridad.
Conocer a que llamamos tiempo en el fondo
Conocer a que llamamos tiempo limite
Conocer a que llamamos profundidad de la immersion
Conocer a que llamamos frontera de seguridad
Conocer cuales son los factores que incrementan el reisgo de la E.D.



Más de 12 Horas => La segunda es simple

Menos de 12 horas y más de 10 minutos => La segunda es sucesiva

Menos de 10 minutos => La segunda es continuada

Puntos a tener en cuenta

Inmersiones Simples

Explicacion
Son aquellas en las que la ultima inmersion se llevo a cabo en mas de 12 horas.
Actuaremos normalmente, como si se tratara de una primera inmersion

Inmersiones Continuadas

Explicacion
Son aquellas que se llevan a cabo en menos de 10 minutos desde la ultima inmersion.
En este caso, la segunda inmersion se considera como continuacion de la primera.

Inmersiones Sucesivas

Explicacion
Aquellas en las que la ultima inmersion se llevo a cabo en mas de 10 minutos pero menos de 12 horas de la anterior

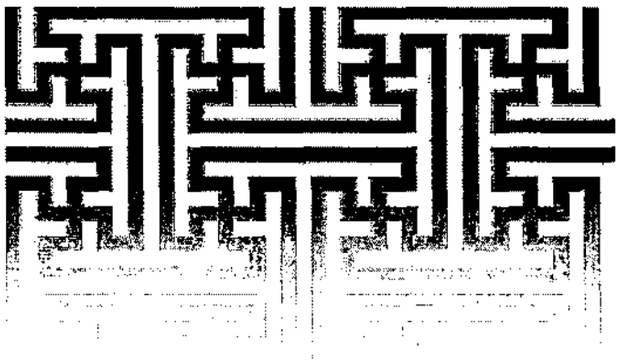
El tiempo limite de una inmersion sucesiva es tl = TL-TNR
Para calcular este tiempo limite (tl) tendremos que calcular el tiempo limite de una inmersion sencilla (TL) a esa profundidad
(Mediante la Tabla del Tiempo Limite y Coeficientes de Salida) y restarle el tiempo de nitrogeno residual (TNR)

Tablas de Inmersion
Tiempo Limite y Coeficiente de Salida
Coeficientes de Nitrogeno Residual

Macias, as Designer you can modify this presentation, please insert the XML file:

XML Monitoring Model:

Send Changes



Bibliografía

A Continuación se detalla la bibliografía utilizada como referencias bibliográficas a lo largo de esta tesis. La lista de publicaciones está compuesta por los trabajos más relevantes del campo de investigación relacionado con el presente trabajo.

[ABG99] Ardissono, L.; Barbero, C.; Goy, A.; Petrone, G.: An Agent Architecture For Personalized Web Stores. Proceedings of the 3trd. International Conference on Autonomous Agents (Agents'99), ACM, May 1999, pp. 182-189.

[ACM00] The Intuitive Beauty of Computer-Human Interaction. Special issue on Programming by Demonstration. Communications of the ACM. March 2000, Vol. 43, No. 3, pp. 73-114.

[AG99] Ardissono, L.; Goy A.: Tailoring the Interaction UIT Users in Electronic Shops. Proceedings of the User Modeling'99 Conference. 1999.

[ASP] Active Server Pages, http://www.asp.net

[BBG+01] Buendía, F.; Benlloch, J.V.; Gómez, J.M.; Burguillo, J.C.; Rodríguez, D.: Herramientas para la Gestión de Recursos Didácticos Accesibles desde Internet, en un Contexto Europeo. Actas del 3er. Simposio Internacional de Informática Educativa. Viseu, Portugal, 26-28 de Septiembre de 2001, pp. 324-333.

[BC98] de Bra, P.; Calvi, L.: AHA! An open Adaptive Hypermedia Architecture. The New Review of Hypermedia and Multimedia, 4. Taylor Graham Publishers, 1998, pp. 115-139.

[BDP+00] Bauer, M.; Dengler, D.; Paul, G.; Myer, M.: Programming by Demonstration for Information Agents. Communications of the ACM, March 2000, Vol. 43, No. 3, pp. 98-103.

[BDP00] Bauer, M.; Dengler, D.; Paul, G.: Instructible Information Agents for Web Mining. Proceedings of the International Conference on Intelligent User Interfaces, January 9-12, 2000. New Orleans, USA, pp. 21-28.

[BEF84] Biggerstaff, T.J.; Endres, D.M.; Forman, I.R.: Table: Object oriented editing of complex structures. Proceedings of the International Conference on Software Engineering. IEEE Computer Society, 1984.

[BES98] Brusilovsky, P.; Eklund, J.; Schwarz, E.: Web-based Education for all: a Tool for the Development of Adaptive Courseware. Proceedings of the Seventh International World Wide Web Conference (WWW'98). April 14-18, 1998. 30 (1-7), pp. 291-300.

[BF98] Benjamins, V.R.; Fensel, D.: Problem-Solving Methods. International Journal of Human-Computer Studies as guest editorial of a special issue on Problem-Solving Methods, 1998, Vol. 49, No. 4, pp. 305-313.

[BFM92] De Baar, D.J.M.J.; Foley, J.D.; Mullet, K.E.: Coupling Applications Design and User Interface Design. In Proceedings of Human Factors in Computing Systems, CHI'92. Monterey, California, May 1992, pp. 259-266.

[BHG+01] Bechhofer, S.; Horrocks, I.; Goble, C.; Stevens, R.: OilEd: a Reasonable Ontology Editor for the Semantic Web. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, 2001, pp 396--408. 2001.

[BHL+94] Bodart, F.; Hennebert, A.M.; Leheureux, J.M.; Provot, I.; Vanderdonckt, J.: A Model-Based Approach to Presentation: A Continum for Task Análisis to Prototype. Proceedings of the 1st. Eurographics Workshop on Design Specification and Verification of Interactive Systems (DSVIS'94). Springer-Verlag, Viena, 1994, pp. 77-94.

[BHL+95] Bodart, F.; Hennebert, A.M.; Leheureux, J.M.; Provot, I.; Sacré, B.; Vanderdonckt, J.: Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. Proceedings of the 2nd. Eurographics Workshop on Design Specification and Verification of Interfactive Systems (DSVIS'95). Springer-Verlag, Viena, 1995, pp. 262-278.

[BHL01] Berners-Lee, T.; Hendler, J.; Lassila, O.: The Semantic Web. Scientific American, May 2001.

[BKV98] Brusilovsky, P.; Kobsa, A.; Vassileva, J. (eds.) Adaptative Hypertext and Hypermedia. Dordrecht, Kluwer Academic Publishers, 1998, pp. 1-43.

[BOS02] Berendt, B.; Otto, A.; Stumme, G.: Towards Semantic Web Mining. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002, pp. 264-278.

[Bra99] de Bra, P.: Design Issues in Adaptive Web-Site Development. Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW, 1999.

[Bru92] Brusilovsky, P.: A Framework for Intelligent Knowledge Sequencing and Task Sequencing. Intelligent Tutoring Systems. Sprinter-Verlag, erlin, 1992, pp. 499-506.

[Bru98] Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. En: Brusilovsky, P., Kobsa, A., Vassileva, J. (eds.): Adaptive Hypertext and Hypermedia. Kluwer Academic Publishers, 1998, pp. 1-43.

[BSW96] Brusilovsky, P.; Schawarz, E.; Weber, G.: ELM-ART: An intelligent

tutoring system on World Wide Web. Proceedings of the Third International Conference on Intelligent Tutoring Systems (ITS'96). Berlin, Springer, pp. 261-269.

[BVE02] Bouillon, L.; Vanderdonckt, J.; Eisenstein, J.: Model-Based Approaches to Reengineering Web Pages. Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design – TAMODIA 2002, 18-19 July, 2002, Bucarest, Romania, pp. 86-95.

[CA99] Cailliau, R.; Ashman, H.: Hypertext in the Web – a History. ACM Computing Surveys. December, 1999, Vol. 31, No. 4es.

[Car01] Carro, R.M.: Un mecanismo basado en tareas y reglas para la creación de sistemas hipermedia adaptativos: aplicación a la educación a través de Internet. Tesis Doctoral. Departamento de Ingeniería Informática, Escuela Politécnica Superior, Universidad Autónoma de Madrid. Abril, 2001.

[CC00] CaldWell, N.; Clarkson, J.: Web-Based Knowledge Management for Distrbuited Design. IEEE Intelligent Systems, May-June 2000, pp. 40-47.

[CDF+00] Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigan, K.; Slattery, S.: Learning to Construct Knowledge Bases from the World Wide Web. Artificial Intelligence, Vol. 118, April 2000, pp. 69-113.

[Che] Chemdex, http://www.chemdex.com

[CHJ02] Cohen, W.W.; Hurst, M.; Jensen, L.S.: A flexible Learning System for Wrapping Tables and Lists in HTML Documents. Proceedings of the WWW'02 Conference. May 7-11, 2002, Honolulu, Hawai, USA.

[CM01a] Castells, P.; Macías, J.A.: Un sistema de presentación dinámica hipermedia para representaciones personalizadas del conocimiento. Libro de Actas del II Congreso Internacional de Interacción Persona-Ordenador (Interacción'01). 16, 17 y 18 de Mayo de 2001, Salamanca (España), pp. 225-234.

[CM01b] Castells, P.; Macías, J.A.: An Adaptive Hypermedia Presentation Modeling System for Custom Knowledge Representations. Proceedings of WebNet'01 - World Conference on the WWW and Internet. Orlando, Florida; October 23-27, 2001. Published by AACE, pp. 148-153.

[CM02a] Castells, P.; Macías, J.A.: Context-Sensitive User Interface Support for Ontology-Based Web Applications. Poster Session of the 1st. International Semantic Web Conference (ISWC'02), Sardinia, Italia; June 9-12th, 2002.

[CM02b] Castells, P.; Macías, J.A.: Un sistema de presentación dinámica en entornos web para representaciones personalizadas del conocimiento. Revista Iberoamericana de Inteligencia Artificial. Número 16, Verano 2002, VI/02, pp. 25-34.

[CMM01] Crescenzi, V.; Mecca, G.; Merialdo, P.: ROADRUNNER: Towards Automatic Data Extraction from Large Web Sites. Proceedings of the 27th VLDB Conference. Roma, Italy, September 11-14, 2001, pp. 109-118.

[Con98] Contreras, J.: Un Système de Génération Automatique de la Composante Tutoriel de Logiciel. Ph.D. Thesis, Université René-Descartes, Paris, 1998.

[CPR99] Carro, R.M.; Pulido, E.; Rodríguez, P.: Dynamic generation of adaptive Internet-based courses. Journal of Network and Computer Applications 22, 1999, pp. 249-257.

[CS99a] Castells, P.; Szekely, P.: Presentations Models by Example. In Design, Specifications and Verifications of Interactive Systems'99. D.J. Duke and A. Puerta (eds.). Springer-Verlag, Viena, 1999, pp. 100-116.

[CS99b] Castells, P.; Szekely, P.: HandsOn: Dynamic Interface Presentation by Example. Proceedings of the HCI International'99. Munich, Gernamy, August 22-26, 1999, pp. 188-1292.

[CSS97] Castells, P.; Szekely, P.; Salcher, E.: Declarative Models of Presentation. Proceedings of the International Conference on Intelligent User Interfaces (IUI'97). January 6-9, 1997, Orlando, Florida, USA, pp. 137-144.

[CTT00] Chen, H.; Tsai, S.; Tsai, J.: Mining tables from large sale HTML texts. Proceedings of the 18th International Conference on Computational Linguistics (COLING), 2000, pp. 166-172.

[Cwe] C-Web, http://cweb.inria.frl

[Cyp93] Cypher A. (ed.).: Watch What I Do: Programming by Demonstration. The MIT Press, 1993.

[DAML] DAML - The DARPA Agent Markup Language Homepage, http://www.daml.org.

[DB96] Douglis, F.; Ball, T.: Tracking and Viewing Changes on the Web. Proceedings of the USENIX Technical Conference, 1996, pp. 156-176.

[DFH+00] Decker, S.; Fensel, D.; van Harmelen, F.; Horrocks, I.; Melnik, S.;

Klein, M.; Broekstra, J.: Knowledge Representation on the Web. Proceedings of the 2000 International Workshop on Description Logics (DL'00). Aachen, Germnay, pp. 89-97, 2000.

[DM00] Domingue, J.; Motta, E.: PlanetOnto: From News Publishing to Integrated Knowledget Management Support. IEEE Intelligent Systems, May-June 2000, pp. 26-31.

[DO01] DAML+OIL, March 2001, http://www.daml.org/2001/03/daml+oil.index

[Dom98] Domingue, J.: Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web. Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop. April 18-23, Banff, Canada, 1998.

[Dub] Dublin Core, http://dublincore.org

[ECJ+99] Embley, D.W.; Campbell, E.M.; Jiang, Y.S.; Liddle, S.W.; Lonsdale, D.W.; Ng, Y.K.; Smith, R.D.: Coneptual-Model-Based Data Extraction from Multiple-Record Web Documents. Data and Knowledge Engineering, Vol. 31, No. 3, November 1999, pp. 227-251.

[Eet94] Roth, S. F. et al: Interactive Graphic Design Using Automatic Presentation Knowledge. CHI'94 Conference. Boston, April 1994, pp. 112-117

[EML] Educational Modeling Language, http://eml.ou.nl

[Enc97a] Encarnação, L.M.: Concept and Realization of Intelligent User Support in Interactive Graphics Applications. Dissertation (Ph.D. Thesis). University of Tübingen, Wilhelm-Schickard-Institute for Computer Science, Germany. July 1997.

[Enc97b] Encarnação, L.M.: Multi-Level user Support through Adaptive Hypermedia: A Highly Application-Independent Help Component. Proceedings of the IUI'97 Conference. Orlando, Florida, USA, 1997, pp. 187-194.

[EP98] Eisenstein, J.; Puerta, A.J.: TIMM: Exploring Task-Interface Links in MOBI-D. Proceedings of the Conference on Human Factors in Computing Systems (CHI'98). ACM Press, 1998.

[FAD+99] Fensel, D.; Angele, J.; Decker S.; Erdmann M.; Schnurr, H. P.; Staab, S.; Studer, R.; Witt, A.: On2broker: Semantic-based access to information sources at the WWW. Actas World Conference on the WWW and Internet (WebNet'99). Honolulu, Hawaii, 1999, pp. 366-371.

[FCM99] Fulantelli, G.; Corrao, R.; Munna, G.: Enhacing User Interaction on

the Web. Proceedings of the WebNet'99, Honolulu, Hawai, USA, October 1999. AACE, pp. 403-408.

[FD] FlipDog web page, http://www.flipdog.com

[Fel99] Fellbaum, C. (ed). WordNet: An Electronic Lexical Database. The MIT Press, 1999.

[FF94] Frank, M.; Foley, J.: A pure reasoning engine for programming by demonstration. Proceedings of the ACM Symposium on User Interface Software and Technology. Marina del Rey, CA, November 2-4, 1994, pp. 95-101.

[FFJ+02] Felfering, A.; Friedrich, G.; Jannach, D.; Zanker, M.: Semantic Configuration Web Services in the CAWICOMS Project. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002, pp. 192-205.

[FFR96] Farquhar, A.; Fikes, R.; Rice, J.: The Ontolingua Server: a Tools for Collaboratorive Ontology Construction. Proceedings of the 10th Knowledge-Based Systems Workshop. Banff, Canada. November 9-14, 1996.

[FHH+00] Fensel, D.; Horrocks, I.; van Harmelen, F.; Decker, S.; Erdmann, M.; Klein, M.: OIL in a nutshell: Knowledge Acquisition, Modeling, and Management. Proceedings of the European Knowledge Acquisition Conference (EKAW-2000). R. Dieng et al. (eds). Lecture Notes in Artificial Intelligence, LNAI, Springer-Verlang, October 2000.

[Fra95] Frank, M.R.: Grizzly Bear: A Demonstrational Learning Tool for a user Interface Specification Language. In Proceedings of the ACM Symposium on User Interface Software and Technology. Pittsburg, Pennsylvania, November 15-17, 1995, pp. 75-76.

[FSF95] Frank, M.; Sukaviriya, P.; Foley, J.: Inference Bear: Designing interactive interfaces through before and after snapshots. Proceedings of the ACM Symposium on Designing Ineractive Systems. Ann Arbor, MI, August 23-25, 1995, pp. 167-175.

[FSL+02] Filha, I.M.E; da Silva, A.S.; Laender, A.H.F.; Embley, D.W.: Using Nested Tables for Representeing and Querying Semistructured Web Data. 14th International Conference on Advanced information Systems Engineering (CaiSE'02). Lecture Notes in Computer Science 2348. Toronto, Ontario, Canada, May 27-31. 2002.

[GBP98] da Graça M.; Benedito, J.; Pontin, R.: Tools for Authoring and Presenting Strcutured Teaching Material in the WWW. Proceedings of WebNet 98 World conference of the WWW, Internet & Intranet, Orlando, florida, November 7-12, 2998, pp. 194-199.

[Gen91] Genesereth, M.R.: Knowledge Interchange Format. Proceedings of the 2nd International Conference on the Principles of Knowledge Representation and Reasoning (KR'91). J. Allen et al. (eds), Morgan Kaufman Publishers, 1991, pp. 238-249.

[GH99] Gilbert, J.E.; Han, C.Y.: Adapting instruction in search of "a significant difference". Journal of Network and Computer Applications, 22, 1999.

[GK95] Green, E.; Krishnamoorthy, M.: Recognition of Tables Using Grammars. Proceedings of the 4th Annual Symposium on Document Analysis and Infomation Retrieval, 1995, pp. 261-278.

[Gru93a] Gruber, T.R.: A translation approach to portable ontologies. Knowledge Acquisition, 5(2), 1993, pp. 199-220.

[Gru93b] Gruber, T. R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N. and Poli, R. (eds.), Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Press, Boston, 1993.

[GRV+98] Gruser, J.R.; Raschid, L.; Vidal, M.E.; Bright, L.: Wrapper Generation for Web Accesible Data Sources. Proceedings of the COOPSIS'98. University of Maryland, Wrapper Generation Project, 1998.

[HBH+98] Horvitz, E.; Breese, J.; Heckerman, D.; Hovel, D.; Rommelse, K.; The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence. Madison, WI, July 1998. Morgan Kaufmann Publishers, pp. 256-265.

[HE01] Han, H.; Elmasri, R.: Architecture of WebOntEx: A System For Semi-Automatic Extraction of Ontologies From Web Pages. Proceedings of the 5th World Multi Conference on Systemics, Cybernetics and Informatics (SCI'01). Orlando, Florida, USA, July 22-25, 2001.

[Hef01] Heflin, J.: Towards the Semantic Web: Knowledge Representation in a Dynamic Distributed Environment. Ph.D. Thesis. University of Maryland, College Park, 2001.

[HGN+97] Hammer, J.; García-Molina, H.; Nestorov, S.; Yernesi, R.; Breuning, M.; Vassalos, V.: Template-based wrappers in the TSIMMIS system. Proceedings of the ACM SIGMOD International Conference on Management of Data. Tucson, Arizona 13-15, May 1997, pp. 532-535.

[HH00] Heflin, J.; Hendler, J.: Semantic Interoperability on the Web. Proceedings of Extreme Markup Languages 2000. Graphic

Communications Association, 2000, pp. 111-120.

[HM00] Hendler, J.; McGuinness, D.L.: The DARPA Agent Markup Language. IEEE Intelligent Systems, Vol. 15, No. 6, November/December 2000, pp. 67-73.

[HMS01] Hand, D.; Mannila, H.; Smyth, P.: Principales of Data Mining. Cambridge, MA. The MIT Press, 2001.

[HMS66] Hunt, E.B.; Marin, J.; Stone, P.J.: Experiments in Induction, Academic Press New York, NY, 1966.

[Hol75] Holland, J. H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Harbor, MI, 1975.

[HP02] Haustein, S; Pleumann, J.: Is Participation in the Semantic Web Too Difficult?. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002, pp. 448-453.

[HS00] Huang, A.W.; Sundaresan, N.: Aurora: A Conceptual Model for Web-Content Adaptation to Support the Universal Usability of Web-based Services. Conference on Universal Usability (CUU). Arlington VA, USA, 2000.

[Huf95] Huffman, S.: Learning information extraction patterns from examples. Workshop on new approaches to learning for natural language processing (IJCAI'95), 1995, pp. 127-142.

[Hur00] Hurst, Matthew Francis: The Interpretation of Tables in Texts. PhD. Thesis. University of Edimburgh, 2000.

[Hyp89] HyperText'89 Proceedings. Pittsburgh, P.A., New York. ACM, Noverber 5-7, 1989

[IAG] Web page of Information Agents' research group, http://www.isi.edu/info-agents

[IMS] Instructional Metadata System, http://www.imsproject.org

[IMS] IMS Global Learning Consortium, Inc, http://www.imsglobal.org

[JDW01] Jin, Y.; Decker, S.; Wiederhold, G.: OntoWebber: Model-Driven Ontology-Based Web Site Management. Semantic Web Working Symposium (SWWS). Stanford University, California, USA. July 30-August 1, 2001.

[Joe99] Joerding T.: A Temporary User Modeling Approach for Adaptive Shopping on the Web. Proceedings of the 2nd. Workshop on Adaptive Systems and User Modeling on the WWW (UM'99). 1999.

[JSP] Sun MicroSystems, Inc.: JavaServer Pages™ Technology. Disponible en la Web: http://java.sun.com/products/jsp.

[JWJ94] Johnson, P.; Wilson, S.; Johnson, H.: Scenarios, Task Análisis, and the ADEPT Design Environment. Scenario-Based Design. J. Carrol (ed.). Addison-Wesley, 1994.

[Kaw] Kawa. Accordit software web page: http://www.accordit.co.uk

[KF03] Klann, M.; Fit, F.: End-User Development. D1.1 Roadmap. Proceedings of the End User Development Workshop at CHI'2003 Conference. Ft. Lauderdale, Florida, USA. April 5-10, 2003.

[KK94] Kay, J.; Kummerfeld, R.J.: An Individualised Course for the C Programming Language. Proceedings of the 2nd International WWW Conference "Mosaic and the Web". 1994.

[KLW95] Kifer, M.; Lausen, G.; Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the ACM, 42, 1995.

[Kop00] Koper, R.: From change to renewal: Educational technology foundations of electronic learning Environments. Educational Technology Expertise Center, Open University of the netherlands, 2000, http://eml.ou.nl

[Kru97] Krulwich, B.: Automaing the Internet: Agents as User Surrogates. IEEE Internet Computing. July/August 1997, Volume 1, No. 4, pp. 34-38.

[KTG00] Kettel, L.; Thomson, J.; Greer, J.: Generating individualized hypermedia applications. Proceedings of the International Workshop on Adaptive and Intelligent Web-based Educational Systems held in Conjunction with ITS 2000. Osnabrück: Technical Report of the Institute for Semantic Information Processing, 2000.

[Küh93] Kühme, T.: A User-Centered Approach to Adaptive Interfaces. Proceedings of the IUI'93 Conference, 1993, pp. 243-245.

[KZ97] Kosbie, D.; Zeiliger, R.: Automating Tasks for groups of Users. Proceedings of the Human-Computer Interfaction Conference INTERACT'97. Sydney, Australia, 1997.

[LFW01] Lieberman, H.; Fry, C.; Weitzman, L.: Explorint the Web with Reconnaissance Agents. Communications of the ACM. August 2001, Vol. 44, No. 8, pp. 69-75.

[LG90] Lenat, D.B.; Guha, R.V.: Building large knowledge-based systems. Representation and inference in the Cyc project. Addison-Wesley. Reading, Massachusets, USA, 1990.

[Lie01] Lieberman, H. (ed): Your Wish is my Command. Programming By Example. Morgan Kaufmann Publishers. Academic Press, USA. 2001.

[Lie99] Lieberman, H.: Integrating User Interface Agents with Conventional Applications. Proceedings of the IUI'99 Conference. Redondo Beach, CA, USA, 1999.

[LN99] Lieberman, H.; Nardi, B.A.: Training Agents to Recognize Text by Example. Proceedings of the Third International Conference on Autonomous Agents. 1999, pp. 116-122.

[LW99] Lau, T.A.; Weld, D.S.: Programming by Demonstration: An Inductive Learning Formulation. Proceedings of the IUI'99 Conference. Redondo Beach, CA, USA, 1999.

[Mac] Macromedia web page: http://www.macromedia.com

[Mau97] Maulsby D.: Inductive Task Modeling for User Interface Customization. Proceedings of the IUI'97 Conference. Orlando, Florida, USA, 1997, pp. 233-236.

[MC00] Macías, J.A.; Castells, P.: Diseño Interactivo de Cursos Adaptativos. 2º Simposio Internacional de Informática Educativa, SIIE'2000. Puertollano (Ciudad Real), del 15-17 Noviembre de 2000.

[MC01b] Macías, J.A.; Castells, P.: A Generic Presentation Modeling System for Adaptive Web-based Instructional Applications. Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'2001), Extended Abstracts. Seattle (Washington), April 2001.

[MC01c] Macías, J.A.; Castells, P.: Adaptive Hypermedia Presentation Modeling for Domain Ontologies. Volume 2 of the Proceedings of HCI International (HCII'01), 9th International Conference on Human-Computer Interaction. New Orleans, Louisiana, USA; August 5-10, 2001. Lawrence Erlbaum Associates, Publishers. London. pp. 710-714.

[MC01d] Macías, J.A.; Castells, P.: Interactive Design of Adaptive Courses. Computer and Education - Towards an Interconnected Society. M. Ortega and J Bravo (eds.). Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 235-242.

[MC01e] Macías, J.A.; Castells, P.: An Authoring Tool for Building Adaptive Learning Guidance Systems on the Web. Active Media Technology. J. Liu et al. (Eds.). Lecture Notes in Computer Science, LNCS 2252. Springer-Verlag Berlin Heidelberg, 2001, pp. 268-278.

[MC02a] Macías, J.A.; Castells, P.: Personalización de Páginas Web

Dinámicas Mediante Ejemplos. Libro de Actas del III Congreso Internacional de Interacción Persona-Ordenador (Interacción'02). 8-10 de Mayo de 2002, Leganés, Madrid (España), pp. 43-50.

[MC02b] Macías, J.A.; Castells, P.: Tailoring Dynamic Ontology-Driven Web Documents by Demonstration. Proceedings of Sixth. International Conference on Information Visualisation (IV'02), July 10-12th, 2002, London, England. IEEE Computer Society Order, Los Alamitos, California, pp. 535-540.

[MC03a] Macías, J.A.; Castells, P.: Dynamic Web Page Authoring by Example Using Ontology-Based Domain Knowledge. Proceedings of the International Conference on Intelligent User Interfaces (IUI'03). Miami, Florida, USA. January 12-15, 2003, pp. 133-140.

[MC03b] Macias, J.A.; Castells, P.: Using Domain Models for Data Characterization in PBE. Proceedings of the End User Development Workshop at CHI'2003 Conference. Ft. Lauderdale, Florida, USA. April 5-10, 2003.

[MC03c] Macías, J.A.; Castells, P.: Detección de Patrones de Iteración en la Edición de Páginas Web Dinámicas Mediante Técnicas de Programación Por Demostración. Por aparecer en las actas del Congreso Interacción Persona-Ordenador (Interacción'03). Vigo, 11-13 de junio de 2003.

[MC03d] Macías, J.A.; Castells, P.: DESK-H: building meaningful histories in an editor of dynamic web pages. To appear in Proceedings of the 11th Internacional Conference on Human-Computer Interaction (HCII). Creta, Grece, June 23-27, 2003.

[McD99] McDaniel, R.: Building Whole Applications Using Only Programming-by-Demonstration. Ph.D. Thesis. Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

[MCH01] Manocha, N.; Cook, D.J.; Holder, L.B.: Structural Web Search Using a Graph-Based Discovery System. intelligence: New Visions of AI in Practice, Vol. 12, Núm. 1, Springtime 2001.

[MCM97] Modugno, F.; Corbett, A.T.; Myers, B.A.: Graphical Representation of Programs in a Demonstrational Visual Shell – An Empirical Evaluation. ACM Transactions on Computer-Human Interaction, September 1997, Vol. 4, No. 3, pp. 276-308.

[MFR+00] McGuinness, D.L.; Fikes, R.; Rice, J.; Wilder, S.: An Environment for Merging and Testing Large Ontologies. Proceedings of the 7th International Conference on Principles of knowledge Representation and Reasoning (KR'00). Breckenridge, Colorado, April 12-15, 2000.

[MGG94] Myers, B. A.; Goldstein, J.; Goldberg, M.: Creating Charts by

Demonstration. Proceedings of the CHI'94 Conference. ACM Press, Boston, April 1994

[Mic] Microsoft Internet Explorer web page: http://www.microsoft.com

[MM00] McDaniel, R.; Myers, B.A.: Gamut: Creating Complete Applications Using Only Programming-by-Demonstration. Submitted to TOCHI (Transactions on Computer-Human Interaction) of the ACM, 1999-2000.

[MM97] Miller, R.; Myers B.: Creating Dynamic World Wide Web Pages By Demonstration. Carnegie Mellon University School of Computer Science, CMU-CS-97-131 and CMU-HCII-97-101, 1997.

[MM99] McDaniel, R.G.; Myers, B.A.: Getting More Out Of Programming-By-Demonstration. Proceedings of the CHI'99 Conference. Pittsburg, PA, USA, 1999.

[MMK93] Myers, B.A.; McDaniel, R.G.; Koshie, D.S.: Marquise: Creating Complete User Interfaces by Demonstration. Proceedings of the INTERCHI'93 Conference, 1993.

[MMM+97] Myers, B.A.; McDaniel, R.G.; Miller, R.C.; Ferrency, A.S.; Faulring, A.; Kyle, B.D.; Mickish, A.; Klimovitski, A.; Doane, P.: The Amulet Environment: New Models for Effective user Interface Software Development. IEEE Transactions on Software Engineering, June, 1997, Vol. 23, No. 6, pp. 347-365.

[MNV02] Missikoff, M.; Navigli, R.; Velardi, P.: The Usale Ontology: An Environment for Building and Assessing a Domain Ontology. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002, pp. 39-53.

[Mob] MOBI-D web site, http://smi-web.stanford.edu/projects/mecano/mobi-d.htm

[Moz] Mozilla web page: http://www.mozilla.org

[MP69] Minsky, M.; Papert, S.: Perceptrons. The MIT Press, Cambridge, MA, 1969.

[MS63] Morgan, J.A.; Sonquist, J.N.: Problems in the Analysis of Survey Data: and a Proposal. Journal of the American Statistical Association", Vol. 58, 1963, pp. 415-434.

[Mur98] Murray, T.: Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. Journal of the Learning Sciences, 1998, Vol 7, No. 1, pp. 5-64.

[Mur99] Murray, T.: Authoring Intelligent Tutoring Systems: An analisis of the state of the art. Journal of Artificial Intelligence in Education,

1999, Vol. 10, pp. 98-129.

[Mus99] Muslea, I.: Extraction Patterns for Information Extraction Tasks: A Survey. Proceedings of AAAI Workshop on Machine Learning for Information Extraction. Orlando, Florida, July, 1999.

[MVR+99] Mayorga, J.I.; Verdejo, F.; Rodríguez-Artacho, M.; Calero, Y.: Domain Modelling to Support Educational Web-Based Authoring. Proceedings of the TET'99 Congress, Norway, June 1999.

[MVW99] Milosavlejevic, M.; Vitali, F.; Watters, C.: Workshop on Virtual Documents, Hypertext Functionality and the Web at the Eighth International World Wide Web conference. toronto, Canada, May 1999.

[MW92] Mo, D.H.; Witten, I.H.: Learning text editing tasks from examples: A Procedural approach. Behaviour & Information Technology, Vol. 11, No. 1, 1992, pp. 32-45.

[MW96] Maulsby, D.; Witten, I.H.: Machine learning in programming by demonstration: lessons learned from Cima. Proceedings of the AAAI Spring Symposium Series: Acquisition, Learning & Demonstration: Automating Tasks for Users, 1996, pp. 66-72.

[Mye88] Myers, B.A.: Creating User Interfaces by Demonstration. Academic Press, San Diego, 1988.

[Mye91] Myers, B.A.: Graphical Techniques in a Spreadsheet for Specifying User Interfaces. Proccedings of the Conference on Human Factors in Computing Systems (SIGCHI'91). ACM Press, New York, 1991, pp. 243-249.

[Mye92] Myers, B.A.: Demonstrational Interfaces: A Step Beyond Manipulation. IEEE Computer. Vol. 25, No. 8, 1992, pp. 61-73.

[Mye93] Myers, B.A.: Why are Human-Computer Interfaces Difficult to Design and Implement?. Carnegie Mellon University School of Computer Science Technical Report, No. CMU-CS-93-183, 1993.

[Mye95] Myers, B.A.: User Interface Software Tools. ACM Transactions on Computer Human Interaction, Vol. 2, No. 1, 1995, pp. 64-103.

[Mye98] Myers, B.A.: Scripting Graphical Applications by Demonstration. Proceedings of the Conference on Human Factors in Computing Systems (CHI'98). ACM Press, 1998.

[Nel87] Nelson, T.H.: Literary Machines. Vers. 87.1. Guide Envelope Document. Bellenuev, WA:OWL International, Inc., 1987.

[Net] Netscape web page: http://www.netscape.com

[Nie89] Nielsen, J.: Hypertext II: Trip Report". June, 1989

[NLK99] Ng, H.T.; Lim, C.Y.; Koo, J.L.T.: Learning to Recognize Tables in Free Text. Proceedings of the 37th Annual Meeting of ACL, 1999, pp. 443-450.

[NO01] Nó, J.; Ortega, S.: Metodología para el Desarrollo de Contenidos en Hipermedia. Actas del 3er. Simposio Internacional de Informática Educativa. Viseu, Portugal, 26-28 de Septiembre de 2001, pp. 17-25.

[NSD+01] Noy, N.F.; Sintek, M.; Decker, S.; Crubezy, M.; Fergerson, R.W.; Musen, M. A.: Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems 16(2):60-71, 2001.

[Ong01] Ong et al. A Web Mining Platform for Enhancing Knowledge Management on the Web. Proceedings of the International Workshop on Integrating Data Mining and Knowledge Management, noviembre 2001.

[Ont] Ontobroker, http://ontobroker.semanticweb.org

[ORB+95] Ortega, M.; Ruiz, F.; Bravo, J.; Ruiz, J.; Domínguez, E.: Hipermedia. Informática educativa: realidad y futuro. Manuel Ortega Cantero [et al.], (editores). – [Cuenca]: Servicio de Publicaciones de la Universidad de Castilla-La Mancha, 1995, pp. 29-50.

[Pat01] Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer Verlag, 2001.

[PBP00] Petrelli, D.; Baggio, D.; Pezzulo, G.: Adaptive Hypertext Design Environments: Putting principles into practice. International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH2000), Tento (Italia), August 28-30, 2000. Lecture Notes in Computer Science LNCS 1892. Peter Brusilovsky, Oliviero Stock, Carlo Strapparava (eds). Springer Verlag, Heidelberg, pp. 202-213.

[PE00] Perkowitz, M.; Etzioni, O.: Towards Adaptive Web Sites: Conceptual Framework and Case Study. Artificial Intelligence, Vol. 118, April 2000, pp. 24-275.

[PE02] Puerta, A.; Eisentein, J.: XIML: A Common Representation for Interaction Data. Proceedings of the Intelligent User Interfaces Conference (IUI'02). San Francisco, California, USA, January 13-16, 2002.

[PE99] Puerta, A.R.; Eisenstein, J.: Towards a General Computational Framework for Model-Based Development Systems. Proceedings of the International Conference on Intelligent User Interfaces (IUI'99).

ACM Press, New York, 1999.

[PF02] Pater-Schneider, P.F.; Fensel, D.: Layering the Semantic Web: Problems and Directions. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002, pp. 16-29.

[PGL95] Pérez, T.A.; Gutiérrez, J.; Lopistéguy, P.: An Adaptive Hypermedia System. Proceedings of the Artificial Intelligence in Education Conference (AIED'95). AACE, Charlottesville, USA, 1995.

[Pia00] Piatetsky-Shapiro, G.: Knowledge Discovery in Databases: 10 years after. SIGKDD Explorations. ACM SIGKDD, Vol. 1, Núm. 2, January 2000, pp. 59-61.

[Pia91] Piatetsky-Shapiro, G.: Knowledge Discovery in Real Databases: A Report on the IJCAI-89 Workshop. AI Magazine, Vol. 11, Núm. 5, January 1991, pp. 68-70.

[PSS01] Paramythis, A.; Savidis, A., Stephanidis C.: AVANTI: a universally accesible web browser. Proceedings of the HCI'01. New Orleans, USA, August 2001, Lawrence Erlbaum Associates, Publishers, pp. 91-95.

[Pue96] Puerta, A.: The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. Proceedings of the Computer-Aided Design of User Interfaces (CADUI'96). 1996, pp. 19-35.

[Pue97] Puerta, A.: A Model-Based Interface Development Environment. IEEE Software, July/August 1997, Vol. 4, No. 14. pp. 40-47.

[PW99a] Paynter, G.W.; Witten, I.H.: Automating Iteration with Programming by Demonstration: Learning the User's Task. Proccedings of the IJCAIWorkshop on Learning about Users, 16th International Joint Conference on Artificial Intelligence. Stockholm, Sweden, 1999.

[PW99b] Paynter, G.W.; Witten, I.H.: Automating iterative tasks with programming by demonstration: a user evaluation. Working paper 99/7. Department of Computer Science. The University of Waikato, New Zealand, 1999.

[RDF] Resource Description Framework, http://www.w3c.org/RDF

[RFP+96] Rice, J.; Farquhar, A.; Piernot, P.; Gruber, T.: Using the Web Instead of a Window System. Proceedings of the CHI'96, Vancouver, B.C. Canada, 1996.

[Ril93] Riloff, E.: automatically constructing a dictionary for information extraction task. Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93), 1993, pp. 811-816.

[ROB95] Ruiz, F.; Ortega, M.; Bravo, J.: Evolución y Perspectivas del Courseware (Cursos realizados por Computador). Informática educativa: realidad y futuro. Manuel Ortega Cantero [et al.], (editores). – [Cuenca]: Servicio de Publicaciones de la Universidad de Castilla-La Mancha, 1995, pp. 51-74.

[RVM+99] Rodríguez-Artacho, M.; Verdejo, M.F.; Mayorga, J.I.; Calero, M.Y.: Using a High-Level Language to Describe and Create Web-Based Learning Scenarios. Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference. November 10-13, 1999. San Juan, Puerto Rico.

[SA00] Sahuguet, A.; Azavant, F.: building Intelligent Web Applications Using Lightweight Wrappers. Data and Knowledge Engineering, 2000.

[SAD+00] Staab, S.; Angele, J.; Decker, S.; Erdmann, M.; Hotho, A.; Maedche, A.; Schmurr, H.-P.; Studer, R.; Sure, Y.: Semantic Community Web Portals. Proceedings of the 9th International WWW Conference. Ámsterdam, May 15-19, 2000.

[SAD01] Santacruz, L.P.; Aedo, I.; Delgado, C.: ELO: Entorno para la generación, Integración y Reutilización de Objetos de Aprendizaje. Actas del 3er. Simposio Internacional de Informática Educativa. Viseu, Portugal, 26-28 de Septiembre de 2001, pp. 293-303.

[SBD03] Schwarzkopf, E.; Bauer, M.; Dengler, D.: Towards Intuitive Interaction ofr End-User Programming. Proceedings of the Intelligent User Interfaces Conference (IUI'03). Miami, Florida, USA, January 12-15, 2003. Poster Session.

[SBF+00] Siekmann, J.; Benzmüller, C.; Fiedler, A.; Franke, A.; Goguadze, G.; Heracek, H.: Kohlhase, M.; Libbrecht, P.; Meier, A.; Melis, E.; Pollet, M.; Sorge, V.; Ullrich, C.; Zimmer, J.; Adaptive Course Generation and Presentation. Proceedings of the International Workshop on Adaptive and Intelligent Web-based Educational Systems held in Conjunction with ITS 2000. Onsabrück: Technical Report of the Institute for Semantic Information Processing, 2000.

[SCC99] Safonov, A.; Constan, J.; Carlis, J.: Towards Web Macros: a Model and a Prototype System for Automating Common Tasks on the Web. Proceedings of the The Future of Web Applications. Human Factors & the Web. June 3, 1999.

[SEM00] Staab, S.; Erdmann, M.; Maedche, A.: An extensible approach for Modeling Ontologies in RDF(S). First ECDL'00 Semantic Web Workshop. Lisbon, Portugal, 2000.

[SG00] Sanrach, C.; Grandbastien, M.: ECSAIWeb: A Web-Based

Authoring System to Create Adaptive Learning Systems. Adaptive Hypermedia and Adaptive Web-Based Systems. Proceedings of the AH2000 Conference. LNCS 1892. Springer-Verlag Berlin Heidelberg, 2000, pp. 214-226.

[SK98] Sugiura, A.; Koseki, Y.: Internet Scrapbook: Automating Web browsing tasks by programming-by-demonstration. Proceedings of the 7th International WWW Conference, Brisbane, Australia, April 1998. Elsevier Science.

[SLN93] Szekely, P.; Luo, P.; Neches, R.: Beyond Interface Builders: Model-Based Interface Tools. Proceedings of the INTERCHI'93 Conference. April, 1993, pp. 383-390.

[SMF+94] Sukaviriya, P.; Muthukumarasamy, J.; Frank, M.; Foley, J.D.: A Model-Based User Interface Architecture: Enhacing a Runtime Environment with Declarative Knowledge, Proceedings of the 1st. Eurographics Workshop on Design Specification and Verification of Interfactive Systems (DSVIS'94). Springer-Verlag, Viena, 1994, 181-197.

[SN98] Szekely, P.; Luo, P.; Neches R.: Beyond Interface Builders: Model-Based Interface Tools. Maybury'98. Section VI: Model-Based Interfaces. 1998, pp. 499-506.

[Sod99] Soderland, S.: Learning information extraction rules for semi-structured and free text. Journal of Machine Learning, Vol. 34, No. 1-3, 1999, pp. 233-272.

[Spy] XMLSpy web page: http://www.xmlspy.com

[SSC+95] Szekely, P.; Sukaviriya, P.; Castells, P.; Muthukumarasamy, J.; Salcher, E.: Declarative interface models for user interface construction tools: the MASTERMIND approach. Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction. Yellowstone Park, USA, August, 1995, pp. 120-149.

[SSD98] Saiz, F.; Szekely, P.; Devang, P.: Customized Web-Based Data Presentation. Proceedings of the WebNet'98. Orlando, Florida, USA, November 7-12, 1998, pp. 792-797.

[SVD+98] da Silva, D.P.; Van Durm, R.; Duval, E.; Olivié, H.: Concepts and Documents for Adaptive Educational Hypermedia: a Model and a Prototype. Actas 2nd Workshop on Adaptive Hypertext and Hypermedia (HYPERTEXT'98). Pittsburg, USA, 1998.

[Sze96] Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. Proceedings of the Computer-Aided Design of User Interfaces (CADUI'96). 1996, pp. xxi-xliv.

[TBP02] Trastour, D.; Bartolini, C.; Preist, C.: Semantic Web Support for the Business-toBusiness E-Commerce Lifecycle. Proceedings of the WWW'02. May 7-11, Honolulu, Hawai, USA, 2002.

[TM98] Tsinakos, A.A.; Margaritis, K.G.: Student Modelling Review – What can be learned for Distance Education?. Proceedings of the WebNet'98. Orlando, Florida, USA, November 7-12, 1998, pp. 1071-1076.

[TW86] Trigg, R.H.; Weiser, M.: Textnet: A Network-based Approach to Text Handling. ACM Transactions on Office Information Systems (TOIS) 4, 1, January 1-23, 1986.

[Vas97] Vassileva, J.: Dynamic Courseware Generation on the WWW. Proceedings 8th World Conference of the AIED Society. Kobe, Japan, 1997, pp. 498-505.

[VB93] Vanderdonckt, J.M.; Bodart, F.: Encapsulating knowledge for Intelligent Automatic Interaction Objets Selection. Proceedings of the Conference on Human Factors in Computing Systems (INTERCHI'93). Addison-Wesley, Ámsterdam, 1993, pp. 424-429.

[VBS01] Vanderdonckt, J.; Bouillon, L.; Souchon, N.: Flexible Reverse Engineering of Web Pages with VAQUISTA. Proceedings of the IEEE 8th Working Conference on Reverse Engineering. Stuttgart, October 2-5, 2001. IEEE Press, pp. 241-248.

[W3C] W3C Consortium. http:://www.w3c.org

[Wan96] Wang, X.: Tabular Abstraction, Editing, and Formatting. PdD Thesis, University of Waterloo, Waterloo, Ontario, Canada, 1996.

[WB90] Wiecha, C.; Boies, S.: Generating user Interfaces: Principles and use of ITS Style Rules. Proceedings of the UIST'90 Conference, 1990, pp. 21-30.

[WBB+90] Wiecha, C.; Bennertt, W.; Boies, S.; Gould, J.; Greene, S.: ITS: A Tool for Rapidly Developing Interactive Applications. ACM Transactions on Information Systems, Vol. 8, No. 3, 1990, pp. 204-236.

[WJ96] Wilson, S.; Johnson, P.: Bridging the Generation Gap: from Work Tasks to User Interface Designs. Proceedings of the Computer-Aided Design of User Interfaces Conference (CADUI'96). Presses Universitaires de Namur, Namur, 1996, pp. 77-94.

[Wol97] Wolber, D.: Pavlov: An Interface Builder for Designing Animated Interfaces. ACM Transactions on Computer-Human Interaction. December 1997, Vol. 4, No. 4, pp. 347-386.

[WS97] Weber G.; Specht M.; User Modeling and Adaptive Navigation Support in WWW-based Tutoring Systems. Proceedings of the User Modeling'97 Conference. Cagliari, Italy, June 2-5, 1997.

[WS99] Wilkinson, R.; Smeaton, A.F.: Automatic Link Generation. ACM Computing Surveys, 31, 4es, December 1999.

[WSC02] Wolber, D.; Su, Y.; Chiang Yih.: Designing Dynamic Web Pages and Persistence in the WYSIWYG Interface. Proceedings of the International Conference on Intelligent User Interfaces (IUI). San Francisco, California, USA. January 13-16, 2002, pp. 228-229.

[XML] Extensible Markup Language, http://www.w3c.org/XML

[XSL] Extensible Stylesheet Language, http://www.w3.org/Style/XSL

[XTM] XML Topic Maps (XTM). Especificación, http://www.topicmaps.org

[Zlo77] Zloff, M.M.: Query-by-Example: A Data Base Language. IBM Systems Journal, Vol. 16, No. 4, pp. 324-343, 1977.

Reunido el tribunal que suscribe en el día
de la fecha acuerda calificar la tesis de
doctoral con: Sobresaliente cum laude por unanimidad
Madrid, 16 de Septiembre de 2002

JESÚS LORÉS

ROBERTO MOLINA SALOTAN

ÓSCAR PASTOR LÓPEZ

ANGEL PUERTA

FRANCISCO SAIZ